Obsah

1 Úvod

1.1 Problém SAT

Problém SAT je zkratka pro „Satisfability problem“. Tím, co se má splnit, je booleovská formule. Splnitelnost booleovských formulí je rozhodovací problém zabývající se vyhodnocováním booleovských výrazů. Uvažujme výrazy, které jsou zapsány pouze pomocí operátorů AND(∧), OR(∨) a NOT(¬), dále obsahují proměnné a závorky. Otázka tedy zní: Existuje k danému výrazu přiřazení, které přiřadí proměnným hodnotu „true“, nebo „false“ tak, aby výraz byl ohodnocen jako „true“?

Ačkoliv se to nezdá, má schopnost řešení problému SAT mnoho praktických aplikací. Mezi nejvýznamnější oblasti využití výsledků řešení splnitelnosti formule patří testování, ověřování a diagnostika VLSI obvodů, plánování a rozvrhování, zpracování počítačového vidění, řízení databází, navrhování integrovaných obvodů a mnoho dalších.

1.2 Booleova logika

Pro porozumění celého textu je potřeba zavést následující pojmy z oblasti matematické logiky:

Booleova logika se skládá z množiny proměnných X= $\{x_1, x_2, x_3 \dots x_n\}$ a základních operátorů (∧ logický součin, ∨ logický součet, ¬ negace, => implikace, <=> ekvivalence).

Výrok je každá věta, u které lze určit pravdivostní hodnotu

Formule výrokové logiky je řetězec(konečná posloupnost) logických proměnných, logických spojek a závorek, jestliže vznikl podle následujících pravidel

- Každá logická proměnná je formule
- Jsou-li α,β formule, pak (¬α) , (α ∧ β), (α ∨ β), (α => β), (α <=> β) jsou také formule

Pravdivostní hodnota formule nabývá dvou hodnot:

- 1 … pravda
- 0 … nepravda

Pravdivostní ohodnocení je zobrazení Z, které každé formuli přiřazuje pravdivostní hodnotu a splňuje následující pravidla:

- (¬α) je pravdivé, pokud α je nepravdivé
- (α ∧ β) je pravdivé, pokud α, β jsou obě pravdivé
- (α ∨ β) je nepravdivé, pokud α, β jsou obě nepravdivé
- (α => β) je nepravdivé, pokud α je pravdivé a β nepravdivé
- (α <=> β) je pravdivé, pokud α, β jsou obě buď pravdivé, nebo nepravdivé

Jestliže ve formuli φ známe pravdivostní hodnoty všech logických proměnných, pak je pravdivostní hodnota φ určena jednoznačně.

Formule s n logickými proměnnými má 2^n různých pravdivostních ohodnocení.

Tautologií nazveme formuli, která je pravdivá ve všech ohodnoceních, např. $(\alpha \vee \neg\alpha)$.
Kontradikcí nazveme formuli, která je nepravdivá ve všech ohodnoceních, např. $(\alpha \wedge \neg\alpha)$.
Splnitelná formule je taková formule, která je pravdivá alespoň v jednom ohodnocení.

Nejjednodušší formulí je literál. Je to proměnná nebo negace proměnné, např. α, $\neg\alpha$.

Každou formuli je možno jednoduchým způsobem převést na formuli obsahující pouze tyto tři operátory: AND($\wedge$), OR($\vee$), NOT($\neg$). Všechny implikace (=>) a ekvivalence (<=>) je možno ze zápisu formule odstranit aplikací následujících pravidel:

- $x_1 \Rightarrow x_2$ je ekvivalentní s $\neg x_1 \vee x_2$
- $x_1 \Leftrightarrow x_2$ je ekvivalentní s $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1)$

O dvou výrazech můžeme říci, že jsou ekvivalentní, pokud mají stejné ohodnocení pro všechny vstupní kombinace proměnných. Příkladem ekvivalencí mohou být např. následující výrazy:

- $\neg\neg x_1 \equiv \neg x_1$
- $\neg x_1 \vee x_2 \equiv x_2 \vee \neg x_1$
- atd.

Minterm je literál nebo součet všech literálů dané klauzule, např. $\neg x_1, \neg x_1 + x_2 + x_3$
Maxterm je literál nebo součin všech literálů dané klauzule, např. $\neg x_1, \neg x_1 \cdot x_2 \cdot x_3$

Každou formuli je možno přepsat do jedné z následujících forem:

- CNF (conjunctive normal form, konjunktivní normálová forma) je maxterm nebo konjunkce konečně mnoha maxtermů, např. $(\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_2$
  Je zřejmé, že formule v CNF je splnitelná, pokud je alespoň jeden člen v každém maxterm splnitelný.
- DNF (disjunctive normal form, disjunktivní normálová forma) je minterm nebo disjunkce konečně mnoha mintermů, např. $(\neg x_1 \wedge x_2 \wedge x_3) \vee \neg x_2$
  Je zřejmé, že formule v DNF je splnitelná, pokud jsou všechny členy v alespoň jednom mintermu splnitelné.

1.3 Složitost problému

Všechny algoritmy můžeme podle jejich složitosti rozdělit do několika tříd:

- třída P = algoritmy s polynomiálně omezenou časovou složitostí
  Rozhodovací problém patří do třídy P, jestliže pro něj existuje algoritmus, který jej řeší v čase $O(n^k)$, kde *n* je velikost instance a *k* je konečné číslo.
- třída NP = takové problémy, pro které existuje nedeterministický algoritmus s polynomiálně omezenou časovou složitostí
- třída PSPACE = problémy, které mohou být vyřešeny pomocí polynomiálně rostoucího množství potřebné paměti, čas řešení není omezen
- třída EXPSPACE = problémy, které mohou být vyřešeny pomocí exponenciálně rostoucího množství potřebné paměti, čas řešení není omezen
- třída EXPTIME = problémy, které mohou být vyřešeny pomocí exponenciálně rostoucího množství potřebného času, dostupná paměť není omezena

Pro všechny výše uvedené třídy problémů platí následující vztah:
$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq EXPSPACE$

NP-úplné problémy jsou nedeterministicky řešitelné v polynomiálně omezeném čase. Jsou to problémy, u kterých je sice velmi obtížné nalézt jejich řešení, ale následné ověření správnosti tohoto řešení je již jednoduché. Pokud existuje nějaký efektivní algoritmus řešící daný NP-úplný problém, existuje také algoritmus řešící všechny NP-úplné problémy polynomiálně převoditelné na daný řešitelný problém.

Více o rozdělení algoritmů viz. [PAA].

Cook (1971) dokázal, že problém splnitelnosti booleovských formulí je NP-úplný.
Speciálním případem obecného problému SAT je problém 3-SAT. Je to taková formule v konjunktivní normálové formě, jejíž každý maxterm je složen právě ze tří termů. Tento problém je jistě NP-úplný, neboť existuje algoritmus s nejvýše polynomiální složitostí, který dovede libovolnou SAT formuli převést na 3-SAT formuli. Tento algoritmus je popsán níže.

1.4 Limity řešitelnosti

V problémech, které vycházejí z praktických příkladů z reálného světa, jsou řešitelné i formule obsahující 100 000 proměnných. Nicméně už pro 10 000 proměnných je nutné zvolit správný algoritmus řešení s vhodnou heuristikou. Řešitelnost uměle generovaných formulí, určených pro testování algoritmů, závisí na poměru počtu klauzulí ku počtu proměnných. O tomto poměru je zmínka v jedné z následujících kapitol.

1.5 Převod problému SAT na problém 3-SAT

V případě řešení obecných problémů většinou dojde k situaci, kdy řešená formule má mnoho klauzulí. Každá klauzule má různý počet proměnných. Řešení takto zadané formule však může být obtížně algoritmizovatelné, proto je mnoho problémů převedeno nejprve na problém SAT, který je dále transformován do takové formy, kdy každá klauzule obsahu stejný počet proměnných (nejčastěji 3). V této podkapitole si ukážeme, jak obecnou formuli SAT v CNF převést na klauzuli se třemi proměnnými (nazývanou také 3-SAT).

Naším úkolem je převést formuli složenou ze součtových termů o různém počtu proměnných na formuli, která se skládá ze součtových termů o přesně třech proměnných. Musíme uvést takový algoritmus, který nezmění řešitelnost celého problému.

Může nastat jeden ze tří následujících případů součtových termů (k – počet proměnných maxtermu):

- $k<3$
- $k=3$
- $k>3$

Označení:
C_j ... seznam proměnných původní klauzule(maxterm)
U_j ... seznam nově přidaných proměnných za účelem transformace

D_j ... nová, již transformovaná klauzule

Pro k=1:
$C_j = \{z\}$
$U_j = \{y_{j1}, y_{j2}\}$
$D_j = \{z, y_{j1}, y_{j2}\}\ \{z, y_{j1}, \neg y_{j2}\}\ \{z, \neg y_{j1}, y_{j2}\}\ \{z, \neg y_{j1}, \neg y_{j2}\}$

Pro k=2:
$C_j = \{z_1, z_2\}$
$U_j = \{y_j\}$
$D_j = \{z_1, z_2, y_j\}\ \{z_1, z_2, \neg y_j\}$

Pro k=3:
požadovaný případ, žádná změna

Pro k>3:
$C_j = \{z_1, z_2, z_3 \dots z_k\}$
$U_j = \{y_{j1}, y_{j2}, y_{jk-3}\}$
$D_j = \{z_1, z_2, y_{j1}\}\ \{\neg y_{j1}, z_3, y_{j2}\}\ \{\neg y_{j2}, z_4, y_{j3}\} \dots \{\neg y_{jk-4}, z_{k-2}, y_{jk-3}\}\{\neg y_{jk-3}, z_{k-1}, z_k\}$

Ve všech případech má původní klauzule C a modifikovaná klauzule D stejnou pravdivostní hodnotu pro stejná přiřazení hodnot $z_1 \dots z_k$.

Dále o převodu SAT na 3-SAT např. zde [Bla].

1.6 Algoritmy řešící SAT

Existuje mnoho různých algoritmů na řešení problému SAT. Tyto algoritmy se od sebe liší svou rychlostí a efektivností, použitím různých heuristik apod.
Všechny algoritmy je možno rozdělit do dvou základních skupin, a to na algoritmy kompletní a nekompletní. Úplné algoritmy dojdou do cíle i v případě, že SAT problém je nesplnitelný. Přibližné algoritmy v případě nesplnitelných výrazů neskončí.

Mezi úplné algoritmy lze zařadit následující:

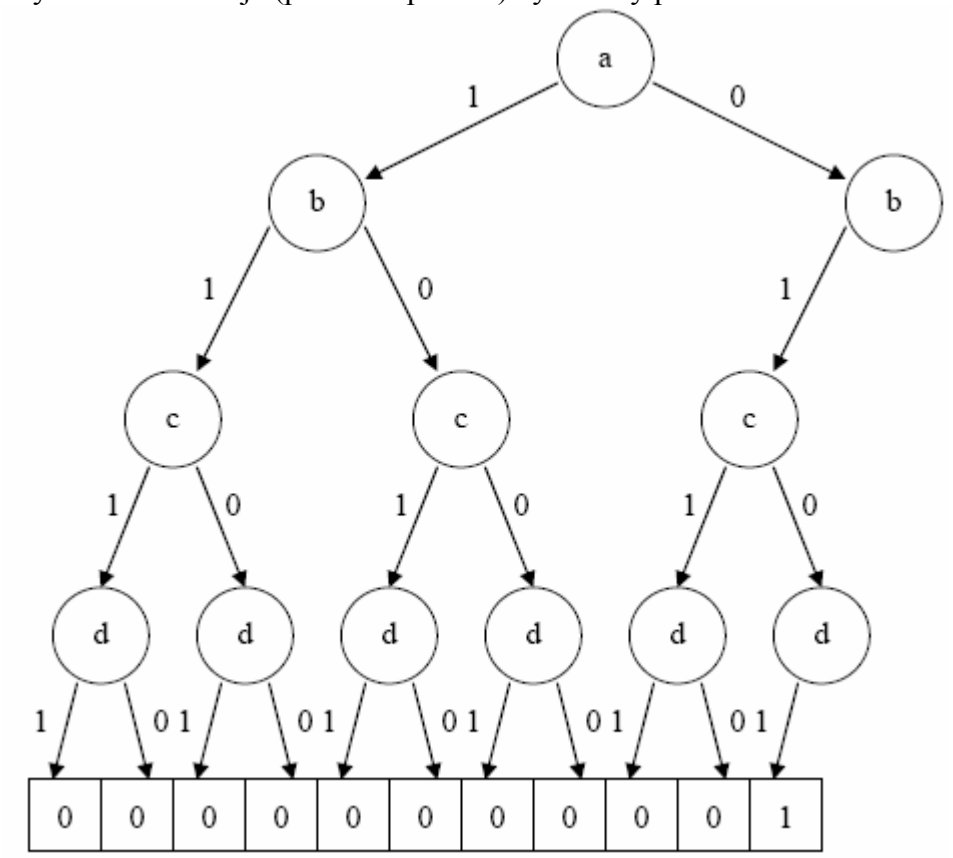
- Truth table
- Prohledávání do hloubky
- Metoda implikace
- Resolution
- Recursive learning

➢ Prohledávání do hloubky

Mějme booleovskou formuli se čtyřmi proměnnými a, b, c, d. Představme si strom o hloubce 4 (počet proměnných) tak, jak je na níže uvedeném obrázku.
Postupným procházením daného stromu do hloubky testujeme všechna možná řešení.
Prohledávání končí nalezením první splnitelné kombinace proměnných.

Máme zadánu následující formuli:

(a ∨ b ∨ c) ∧ (a ∨ b ∨ ¬c) ∧ (¬a ∨ b ∨ ¬c) ∧ (a ∨ c ∨ d) ∧ (¬a ∨ c ∨ d) ∧ (¬a ∨ c ∨ ¬d) ∧ (¬b ∨ ¬c ∨ ¬d) ∧ (¬b ∨ ¬c ∨ d)
Níže uvedený obrázek ukazuje (pro tento příklad) vytvořený prohledávací strom.



Obr. 1 - Průběh prohledávání stavového prostoru metodou prohledávání do hloubky

Z výše uvedeného obrázku je vidět, že formule je splnitelná pro ohodnocení a=0, b=1, c=0, d=1.

Tuto metodu je možno dále vylepšit pomocí heuristiky, které umožňují rozpoznat nesplnitelnost dané větve ještě dříve, než bude prohledána až do konce.
Jako příklad bychom mohli uvést heuristiku využití implikace (viz. [Kut]). Pokud v průběhu rozhodování vznikne klauzule, která obsahuje pouze jeden neohodnocený literál, a ostatní literály v klauzuli jsou ohodnoceny jako nesplnitelné, je ohodnocení tohoto literálu implikováno právě těmito již ohodnocenými literály, tj. ten literál, který jako poslední zůstává stále ještě neohodnocen, ovlivní, zda celá klauzule bude splněna nebo ne.

Mezi přibližné algoritmy lze zařadit následující:

- Lokální metody (GSAT, WalkSat ...)
- Simulované ochlazování
- Tabu search
- Genetické algoritmy

➢ Lokální metody

Všechny lokální metody jsou založeny na následujícím rámci:

T = náhodné pravdivostní ohodnocení formule
F = celkový počet překlopení proměnných (jak dlouho má hledat řešení. Pokud řešení nenalezne, je formule považována za nesplnitelnou)
FOR (j=1 to F)
{
    IF (T splňuje funkční předpis){return T;}
    V = vyber proměnnou pomocí nějaké z níže uvedených heuristik
    T´= pravdivostní ohodnocení, které vznikne z T invertováním proměnné V
}
return failure ... žádné ohodnocení, při kterém je formule splnitelná, nenalezeno

V souvislosti s algoritmy typu lokální metody zavádíme následující pojmy:
Počet všech klauzulí (splněných i nesplněných) v CNF je P.
Nechť T je ohodnocení formule F v CNF formě. B_0 je počet klauzulí, které jsou nesplněné v T. Splněných klauzulí v ohodnocení F je P- B_0 .
Nechť T´ je ohodnocení formule F v CNF formě takové, že proměnná V je invertována. B_1 je počet klauzulí, které jsou nesplněné v T´. Splněných klauzulí v ohodnocení F je P- B_1 .

Čistý zisk (net gain) proměnné V je hodnota $B_1 - B_0$.
Záporný zisk (negative gain) proměnné V je hodnota (P- B_0) - B_1.
Kladný zisk (positive gain) proměnné V je hodnota $B_0 - (P - B_1)$.
Stáří proměnné(Variable Age) V je počet překlopení(invertování) libovolných proměnných od doby, kdy byla naposledy hodnota proměnné V invertována.

Následuje výčet a stručný popis některých významných heuristik, které napomáhají výběru proměnné, která má být překlopena:

- **GSAT** [Selman, Mitchell, Levesque 1992]
  K překlopení proměnné vyber tu proměnnou, která má nejvyšší čistý zisk.
  Ukonči se náhodně. Metoda provádí hladové tahy.

- **HSAT** [Gent and Walsh, 1993]
  Stejné jako GSAT, ale ukončení prohledávání závisí na nejvyšším stáří proměnné.
  Metoda provádí hladové tahy.

- **GWSAT** [Selman, Kautz 1993]
  S pravděpodobností p vybere proměnnou z náhodně vybrané nesplněné klauzule (broken clause), jinak se chová stejně jako GSAT.

- **WalkSAT** [Selman, Kautz, Cohen 1994]
  Vybere náhodně jednu nesplněnou klauzuli (broken clause - BC). Jestliže mají některé proměnné v BC záporný zisk roven 0, pak vyber k překlopení náhodně jednu z nich. Jinak vyber s pravděpodobností p nějakou proměnnou z BC k překlopení a s pravděpodobností (1 - p) vyber proměnnou s minimálním záporným ziskem
  Ukonči se náhodně.

- **Novelty** [McAllester, Selman, Kautz 1997]
  Vyber náhodně jednu nesplněnou klauzuli BC. Vyber tu proměnnou V z BC, která má maximální čistý zisk, pokud zároveň tato proměnná nemá minimální stáří (variable age)

ze všech proměnných v dané klauzuli. Ve druhém případě vyber proměnnou V s pravděpodobností (1-p); jinak překlop V_2 s druhým největším čistým ziskem.

- **Novelty+** [Hoos and Stutzle 2000]
  Stejné jako Novelty, ale poté, co je BC vybrána s pravděpodobností p, vyber náhodně jednu proměnnou z BC; jinak pokračuj stejně jako algoritmus Novelty.

- **R-Novelty** [McAllester, Selman, Kautz 1997] and **R-Novelty+** [Hoos and Stutzle 2000]
  Další možné heuristiky, které jsou podobné heuristikám Novelty/Novelty+, avšak jsou mnohem složitější.

Více informací viz. např. [Fuk].

➢ Simulované ochlazování

Simulované ochlazování je technika pro řešení optimalizačních problémů, hledající dobrou aproximaci pro globální optimum zadané funkce na velkém prohledávaném prostoru. Jméno a inspirace této techniky pochází ze žíhání(chlazení) v hutnictví – technika zahrnuje zahřívání a kontrolované ochlazování materiálu za účelem zvýšení velikosti jeho krystalů a snížení počtu jeho defektů. Dodání tepla způsobí odtržení atomů z jejich původního místa (lokální minimum vnitřní energie). Uvolněné atomy se poté náhodně pohybují mezi stavy s vyšší energií. Pomalé ochlazování jim dává větší šanci získat konfiguraci s nižší vnitřní energií, než měly v původním místě před odtržením.

Analogicky s tímto fyzikálním procesem je následující: každý krok algoritmu simulovaného ochlazovnání nahradí algoritmus aktuální řešení náhodně vybraným „blízkým" řešením. „Blízké" řešení je vybráno s pravděpodobností, která závisí na rozdílu mezi odpovídajícími funkčními hodnotami a na globálním parametru T (nazýván teplota), který je průběžně snižován. Závislost je přibližně taková, že aktuální řešení je měněno téměř náhodně, když hodnota T je velká, ale hodnota řešení rychle klesá, když se hodnota T blíží k 0. V průběhu hledání optimálního řešení je dovoleno zvýšit hodnotu aktuálního řešení – to chrání algoritmus před uváznutím v místě lokálního minima.

Následuje pseudokód názorně popisující průběh algoritmu:

```
s := s0; e := E(s)                                  // Počáteční stav, energie
k := 0                                              // Počítadlo – kolik kroků jsme udělali od začátku
                                                    výpočtu
while k < kmax and e > emax                         // Algoritmus prováděj do té doby, dokud není
                                                    překročen maximální počet kroku a řešení není
                                                    dostatečně dobré
  sn := neighbour(s)                                // Vyber nějaké "blízké" řešení
  en := E(sn)                                       // Vypočítej hodnotu tohoto řešení
  if random() < P(e, en, temp(k/kmax)) then         // Máme se přesunout na dané řešení?
    s := sn; e := en                                // Ano, změň stav
  k := k + 1                                        // Zvyš hodnotu počítadla o 1
return s                                            // Vrať aktuální řešení
```

➢ Tabu search

Tabu search je pokročilá technika pro místní vyhledávání (local search) využívající paměť, aby došlo k vyhnutí se místnímu(lokálnímu) maximu. Princip celé metody je velmi jednoduchý: v každé iteraci provede nejlepší možný tah (pod pojmem tah si lze představit např. překlopení určité proměnné). Pokud je daná konfigurace jednou navštívena, stává se tabu. To znamená, že algoritmu není dovoleno navštívit tuto konfiguraci po předem daný počet iterací. Zapamatování si posledních konfigurací může být nákladné. To ovšem závisí na délce řetězce reprezentujícího instanci řešené úlohy. Jednodušší (co se paměťové náročnosti týče) je ukládat do paměti pouze indexy proměnných, které byly naposledy invertovány. Tah je proveden pouze tehdy, pokud je nejlepší možný a pokud není tabu. Tah je po prvním zavolání (spuštění) uložen do tabu seznamu – realizovaného jako fifo fronta s pevnou délkou a je z této fronty odstaněn až po uplynutí λ iterací.

2 Problémy řešitelné pomocí SAT

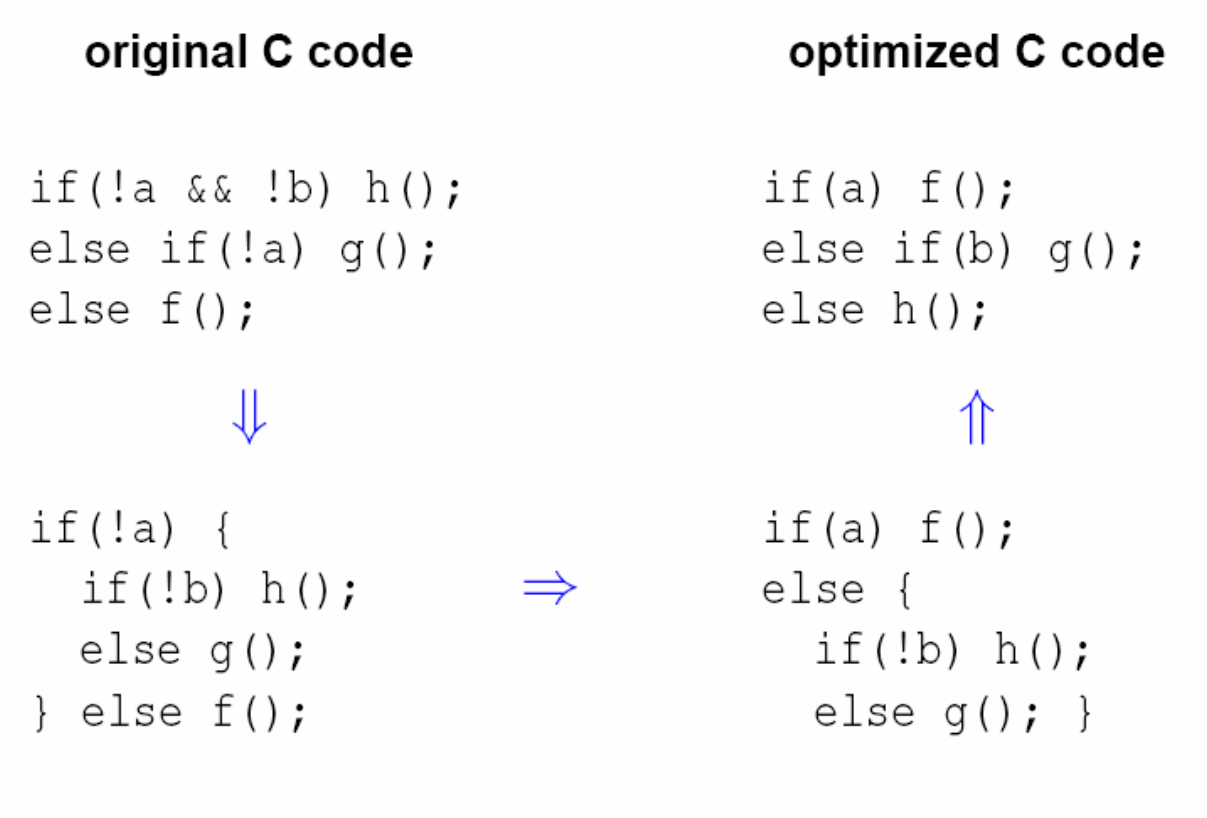
2.1 Úvod

V této kapitole si ukážeme některé praktické i teoretické problémy, které mohou být převedeny na SAT a následně vyřešeny. Zaměříme se pouze na NP úplné problémy. Každý problém obsahuje algoritmus (polynomiálně složitý) pro převedení na SAT. Velmi důležitá je i reprezentace zápisu a řešení daného problému.

2.2 Ověřování shodnosti

Máme zadaný kód v nějakém programovacím jazyku. Překladač tento kód nějakým způsobem optimalizuje. Chceme zjistit, jestli je původní i optimalizovaný kód rovnocenný, tj. zda ve všech případech dává původní i optimalizovaný kód stejný výsledek. Na vyřešení této úlohy můžeme s výhodou využít problém SAT. Viz. např. [Kro]. Celý postup si ukážeme na níže uvedeném příkladu.

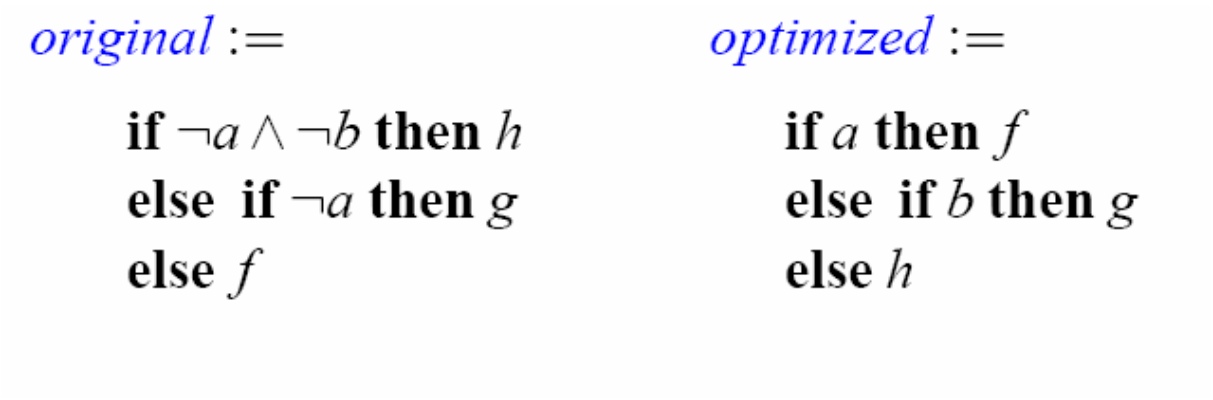
Optimalizace IF-THEN-ELSE podmínkového příkazu



Obr. 2 – Ukázka ekvivalentních výrazů z jazyka C - 1

Jak zjistit, že obě verze kódu v jazyku C jsou rovnocenné?

- pojmenujeme všechny proměnné a procedury jako nezávisle booleovské proměnné



Obr. 3 – Ukázka ekvivalentních výrazů z jazyka C - 2

- převedeme IF-THEN-ELSE příkaz na booleovskou formuli

Zavedeme funkci PŘEVEĎ, která transformuje příkaz na booleovskou formuli.

PŘEVEĎ(if **x** then **y** else **z**) = (**x** ∧ **y**) ∨ (¬**x** ∧ **z**)

Buď platí x a provede se y nebo neplatí x a provede se z.

- zjistíme, zda jsou obě formule ekvivalentní

PŘEVEĎ(původní formule) <=> PŘEVEĎ(optimalizovaná formule)

- Abychom mohli celý problém řešit pomocí SAT, přetransformujeme celé zadání následujícím způsobem:

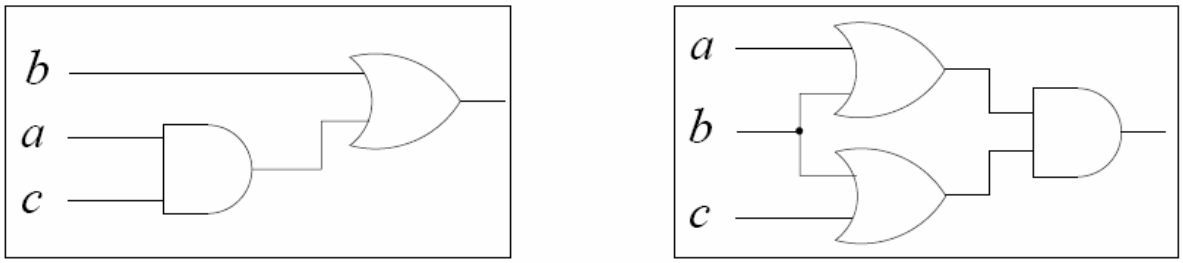
Existuje nějaké ohodnocení proměnných a, b, f, g, h takové, že booleovská formule

PŘEVEĎ(původní) != PŘEVEĎ(optimalizovaná)?

Dvojice symbolů „!=“ označuje operátor „není ekvivalence“.

Pokud se nám nyní povede nalézt ohodnocení, kdy je takto vytvořená formule splnitelná, tak víme, že oba podmíněné příkazy nejsou rovnocenné(existuje takové ohodnocení, kdy původní podmínka je splnitelná a optimalizovaná podmínka je nesplnitelná nebo naopak).

Ověření, zda jsou dva kombinační obvody ekvivalentní



Obr. 4 - Dvojice ekvivalentních logických obvodů

Stejný problém, jako v případě shodnosti dvou podmínek v jazyku C, je i následující problém – shodnost dvou kombinačních logických obvodů. Ačkoliv jsou oba obvody odlišné co se počtu hradel, jejich topologie a spojení týče, jsou po funkční stránce zcela identické.
Funkci kombinačního obvodu na obrázku vlevo můžeme popsat vztahem (1), funkci pravého kombinačního obvodu popíšeme vztahem (2).

(1) $\mathbf{b \vee (a \wedge c)}$

(2) $\mathbf{(a \vee b) \wedge (b \vee c)}$

Opět hledáme ohodnocení proměnných a, b, c takové, že formule b ∨ (a ∧ c) != (a ∨ b) ∧ (b ∨ c) je splněná. Pokud řešení nenalezneme, tak jsou oba kombinační obvody po funkční stránce totožné, v opačném případě jsou oba obvody po funkční stránce různé.

2.3 ATPG

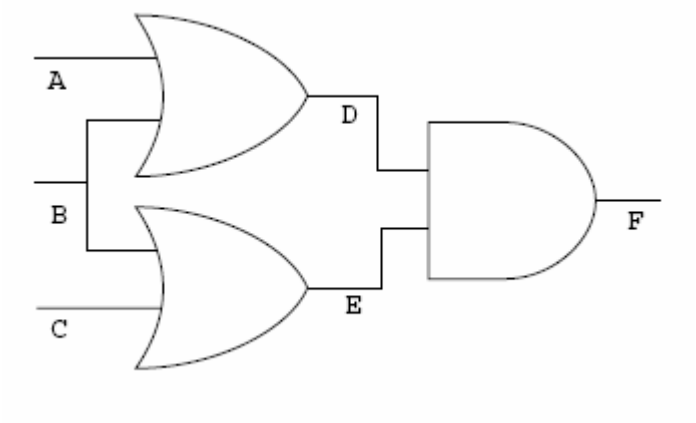
ATPG(Automatic Test Pattern Generation) je automatické generování testovacích vzoru pro kombinační a sekvenční číslicové obvody.

Vzhledem k rozsahu moderních číslicových obvodů není snadné (a prakticky ani možné) vytvořit rozhraní, pomocí kterého by se daly otestovat všechny funkce obvodu. Cílem výrobců je vyrábět s co největším ziskem, tedy co nejrychleji a nejlevněji. To, jak bude číslicový obvod levný, závisí na skutečnosti, kolik jich vyrobíme správně funkčních. Odhalit poruchu součástky co nejdříve je vždy nejlevnější.

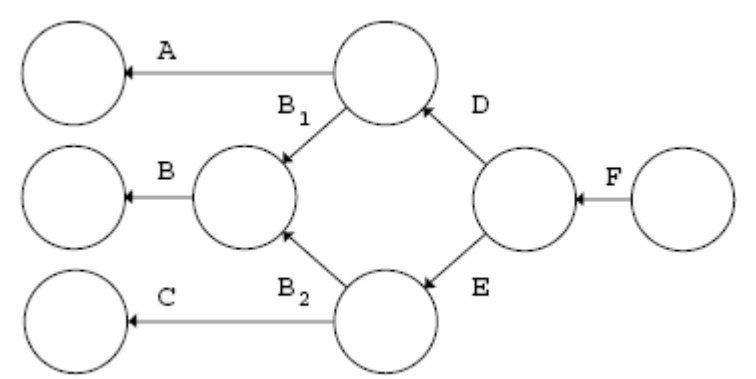
U číslicových obvodů uvažujeme dvě základní poruchy, a to trvalou jedničku(Sa1 = Stuck-At 1) a trvalou nulu(Sa0 = Stuck-At 0). Omezíme se tedy na hledání pouze těchto dvou druhů poruch. Je dobré si uvědomit, že téměř všechny ostatní formy poruch (zkraty, přemostění, přerušení apod.) jsou shodné s poruchami Sa0 nebo Sa1.

ATPG generuje množinu testovacích vektorů pro kombinační obvod.

Abychom mohli vygenerovat testovací vektor pro jednu poruchu typu Sa0 nebo Sa1, vygenerujeme nejprve formuli, která definuje množinu testovacích vektorů detekujících poruchu, a pak užijeme algoritmu pro výpočet SAT a hledáme řešení. Uveďme si vše na jednoduchém příkladu. Orientovaný acyklický graf reprezentuje vlastnosti našeho ukázkového obvodu. Uzly grafu jsou vstupy obvodu, výstupy, hradla a všechna místa, kde dochází k větvení. Hrany potom reprezentují spojení mezi jednotlivými prvky obvodu. Každé hraně je přiřazena proměnná. Viz. dva níže uvedené obrázky.

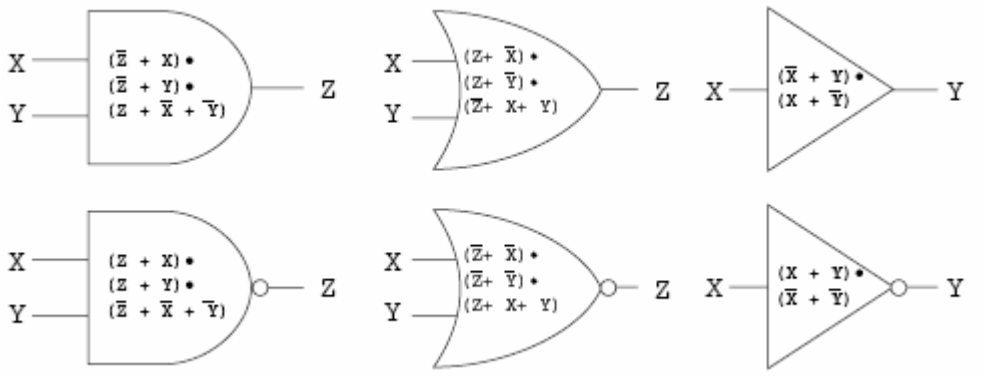


Obr. 5 - Vzorový obvod



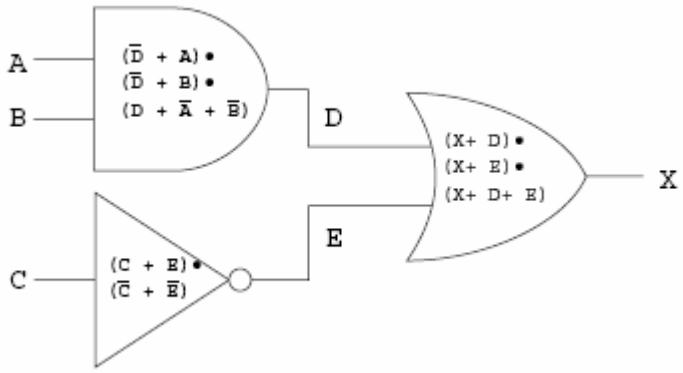
Obr. 6 - Orientovaný acyklický graf reprezentující topologii výše uvedeného obvodu

Každý uzel orientovaného grafu je popsán výrazem, který reprezentuje operaci, ke které dojde na hradlu nebo v bodě větvení. Například invertor se vstupem X a výstupem Y bude označen jako Y = ¬X; AND hradlo se vstupy X a Y a výstupem Z bude označeno výrazem Z = X * Y. Každý uzel je označen výrazem, který obsahuje pouze proměnné pro vstupní a výstupní hrany.



Obr. 7 - CNF formule pro základní hradla

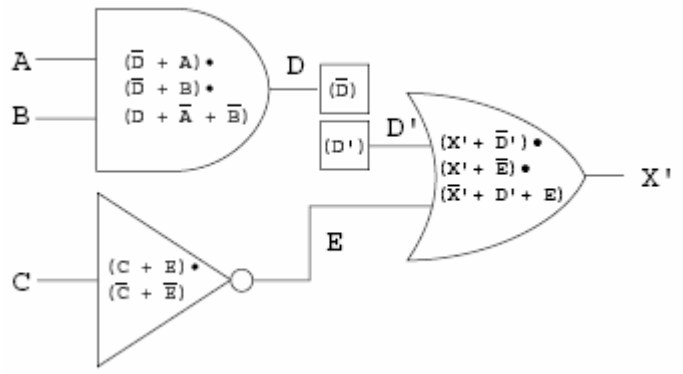
Vzhledem k tomu, že každé hradlo a bod větvení je popsán formulí, která musí být nezávisle splněná (nezávisle na ostatních uzlech a jejich vstupech a výstupech), vytvoříme formuli pro celý obvod následujícím způsobem. Začneme na výstupu a procházíme celý graf, vytvoříme konjunkci klauzulí všech uzlů, které navštívíme. Za předpokladu, že formule pro každý uzel grafu bude nezávisle splnitelná, musí být celá formule splnitelná. Níže uvedený obrázek ukazuje náš ukázkový obvod a jeho charakteristickou formuli.



Obr. 8 - Obrázek za účelem vygenerování CNF formule pro správně fungující obvod

Konečný výraz reprezentující výše uvedený obrázek:
(X + ¬D) * (X + ¬E) * (¬X + D + E) * (¬D + A) * (¬D + B) * (D + ¬A + ¬B) * (C + E) * (¬C + ¬E)

Nesprávně fungující obvod můžeme vytvořit tak, že nejprve zkopírujeme původní (správně fungující) obvod, přejmenujeme proměnné a vložíme dva nové uzly reprezentující předpokládané místo, kde dojde k poruše Sa0 nebo Sa1. To znamená, že pokud obvod obsahuje poruchu, kterou máme v plánu detekovat, tak na jednom konci spoje bude generována (a dále propagována) jedna hodnota a na druhém konci bude jiná hodnota. Oba nově vytvořené uzly označíme hodnotami D a D', které reprezentují chování obvodu v místě poruchy.



Obr. 9 - Obrázek za účelem vygenerování CNF formule pro špatně fungující obvod, na spoji D je trvalá 1

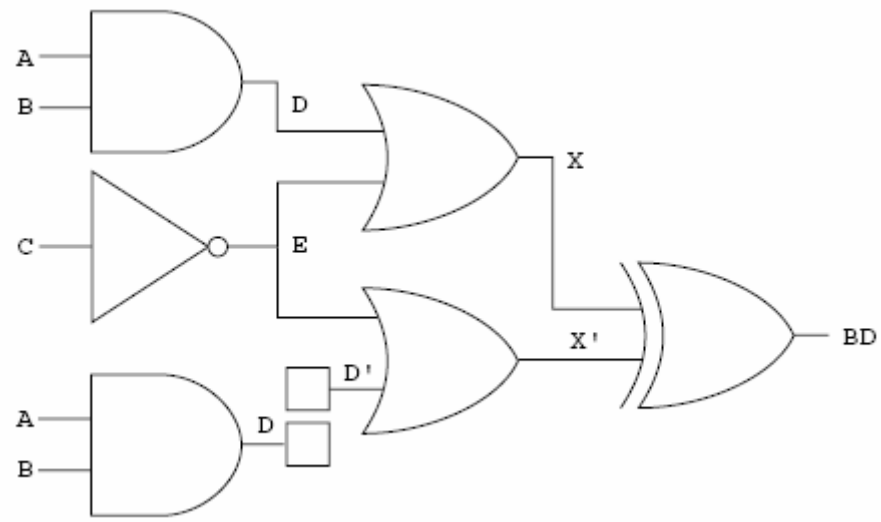
Protože správně pracující i chybný obvod mají zcela shodné chování s výjimkou uzlů, které jsou ovlivněny výskytem poruchy, pouze spoje, které se vyskytují mezi místem výskytu chyby a výstupem, musí být přejmenovány.

Formuli pro chybný obvod vytvoříme stejným způsobem, jako jsme vytvářeli formuli pro správně pracující obvod.

Konečný výraz reprezentující výše uvedený obrázek:
(X' + ¬D') * (X‘ + ¬E') * (¬X‘ + D' + E) * (D‘) * (C + E) * (¬C + ¬E)

Klauzuli (¬D‘) nezahrnujeme, protože v místě chyby je obvod přerušen.

Abychom otestovali danou poruchu, potřebujeme najít množinu vstupů, které způsobí, že se výstupní hodnoty bezchybného a chybného obvodu budou lišit. Pokud vytvoříme konjunkci dříve získaných CNF forem bezchybného a chybného obvodu a pokup jejich výstupy připojíme na vstup XORu, dostaneme formuli popisující všechny možné testovací vektory. Obvod vytvořen v souladu s naším postupem je uveden níže.



Obr. 10 - Obrázek za účelem vygenerování CNF formule pro porovnání výstupu správně a špatně fungujícího obvodu

Konečný výraz reprezentující výše uvedený obrázek:
(X + ¬D) * (X + ¬E) * (¬X + D + E) * (¬D + A) * (¬D + B) * (D + ¬A + ¬B) (X' + ¬D') *
(X‘ + ¬E') * (¬X‘ + D' + E) * (D‘) * (C + E) * (¬C + ¬E) *
(¬X + X' + BD) * (X + ¬X' + BD) * (¬X + ¬X' + ¬BD) * (X + X' + ¬BD) ,
kde první řádek reprezentuje špatný obvod, druhý řádek reprezentuje správně fungující obvod a třetí řádek popisuje koncové hradlo XOR.

Všimněme si, že v obou popisech obvodu se vyskytovaly stejné klauzule, konkrétně (C + E) a (¬C + ¬E). Tyto klauzule se ve výsledném zápisu vyskytují pouze jednou., neboť AND je idempotentní.

Pozn. O operaci prohlásíme, že je idempotentní, pokud vrací stejný výsledek bez ohledu na to, zda byla na stejný prvek aplikována jednou nebo vícekrát. V souvislosti s hradlem (s mnoha vstupy) AND se jedná o to, že pokud na více vstupů je připojen stejný obvod, tak výsledek bude stejný, jako kdyby tento vstup byl připojen pouze jedenkrát na jeden vstup.

Pokud se nám nyní povede nalézt takové ohodnocení vstupů, kdy formule je splnitelná, je výstup obvodu 1 a nalezli jsme testovací vektor pro hledaný typ chyby v daném místě.

Dále k tomuto tématu viz. [Lar] nebo [Sta].

2.4 Barvení grafu k-barvami

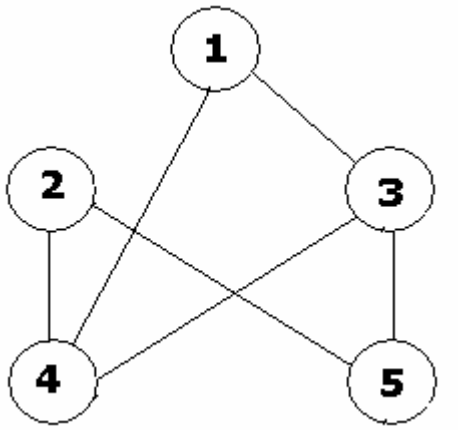
Problém barvení grafu je definován následujícím způsobem: Je dán neorientovaný graf G(V, E) a přirozené číslo k. Otázka zní: Existuje nějaké přiřazení barev uzlům takové, že

žádné dva uzly se stejnou barvou nejsou přímo spojeny žádnou hranou. Pokud je to možné, řekneme, že graf je k-chromatický.

- Zakódování problému
  V tomto případě je nejlepší, když zavedeme k.|V| proměnných $x_{i,j}$. i nabývá hodnot 1..|V|, j nabývá hodnot 1..k. Pokud $x_{i,j}$ nabývá hodnoty pravda, tak to znamená, že vrchol s číslem i je obarven barvou j.
- Alespoň jedna barva
  Chceme, aby každému vrcholu byla přiřazena alespoň jedna barva. Tato podmínka generuje jednu klauzuli pro každý uzel, celkem generuje |V| klauzulí. Klauzule pro uzel v_i vypadá následovně:
  $x_{i,1} \vee x_{i,2} \vee x_{i,3} \ldots \vee x_{i,k}$
- Nejvýše jedna barva
  Bez zavedení tohoto pravidla by mohlo dojít k situaci, kdy by mohl SAT solver dodat řešení, kdy by jednomu uzlu mohl přiřadit více než jednu barvu. Abychom tomu zabránili, musíme dodat další pravidlo, které zajistí, že každý uzel bude mít přiřazenu právě jednu barvu. Tuto podmínku můžeme zajistit tak, že pokud bude uzel v_i obarven barvou c, pak nemůže být obarven jinou barvou d, d!=c. Zápis vypadá následovně:
  $(x_{i,c} \Rightarrow \neg x_{i,d}) \Leftrightarrow (\neg x_{i,c} \vee \neg x_{i,d})$, $1 <= c < k$, $c+1 <= d <= k$
  Pro k barev máme k(k-1)/2 dvojic barev. Pro celý graf tak dostaneme celkem |V|.k.(k-1)/2 klauzulí.
- Různé barvy
  Toto je nejdůležitější pravidlo, které zajišťuje, že dva vrcholy spojené hranou jsou obarveny různou barvou. Pro každou hranu $E = \{v_i, v_j\}$ platí: Jestliže v_i je obarven barvou c, pak v_j nesmí být obarven barvou c. Zápis tohoto pravidla vypadá následovně:
  $(x_{i,c} \Rightarrow \neg x_{j,c}) \Leftrightarrow (\neg x_{i,c} \vee \neg x_{j,c})$, $1 <= c <= k$
  Pro každou hranu musíme vytvořit k takových klauzulí, jednu pro každou barvu. Pro celý graf tak dostaneme celkem k.|E| klauzulí.

Shrnutí: všechna tři výše uvedená pravidla vytváří celkem |V| + |V|.k.(k-1)/2 + k.|E| klauzulí.

Celý převod si ještě ukážeme pro jednoduchý konkrétní příklad: graf s 5 uzly a 6 hranami chceme obarvit třemi barvami.



Obr. 11 - Jednoduchý graf, který se budeme snažit obarvit třemi barvami

Podle prvního pravidla(každý z pěti uzlů je obarven alespoň jednou ze tří barev) vytvoříme následující klauzule:
$(x_{1,1} \vee x_{1,2} \vee x_{1,3}) \wedge (x_{2,1} \vee x_{2,2} \vee x_{2,3}) \wedge (x_{3,1} \vee x_{3,2} \vee x_{3,3}) \wedge$
$(x_{4,1} \vee x_{4,2} \vee x_{4,3}) \wedge (x_{5,1} \vee x_{5,2} \vee x_{5,3})$

Podle druhého pravidla(každý z pěti uzlů je obarven nejvýše jednou ze tří barev) vytvoříme následující klauzule:
$(\neg x_{1,1} \vee \neg x_{1,2}) \wedge (\neg x_{1,1} \vee \neg x_{1,3}) \wedge (\neg x_{1,2} \vee \neg x_{1,3}) \wedge$
$(\neg x_{2,1} \vee \neg x_{2,2}) \wedge (\neg x_{2,1} \vee \neg x_{2,3}) \wedge (\neg x_{2,2} \vee \neg x_{2,3}) \wedge$
$(\neg x_{3,1} \vee \neg x_{3,2}) \wedge (\neg x_{3,1} \vee \neg x_{3,3}) \wedge (\neg x_{3,2} \vee \neg x_{3,3}) \wedge$
$(\neg x_{4,1} \vee \neg x_{4,2}) \wedge (\neg x_{4,1} \vee \neg x_{4,3}) \wedge (\neg x_{4,2} \vee \neg x_{4,3}) \wedge$
$(\neg x_{5,1} \vee \neg x_{5,2}) \wedge (\neg x_{5,1} \vee \neg x_{5,3}) \wedge (\neg x_{5,2} \vee \neg x_{5,3})$

Podle třetího pravidla(každá hrana spojuje pouze různě obarvené uzly) vytvoříme následující klauzule:
$(\neg x_{1,1} \vee \neg x_{3,1}) \wedge (\neg x_{1,2} \vee \neg x_{3,2}) \wedge (\neg x_{1,3} \vee \neg x_{3,3}) \wedge$
$(\neg x_{1,1} \vee \neg x_{4,1}) \wedge (\neg x_{1,2} \vee \neg x_{4,2}) \wedge (\neg x_{1,3} \vee \neg x_{4,3}) \wedge$
$(\neg x_{2,1} \vee \neg x_{4,1}) \wedge (\neg x_{2,2} \vee \neg x_{4,2}) \wedge (\neg x_{2,3} \vee \neg x_{4,3}) \wedge$
$(\neg x_{2,1} \vee \neg x_{5,1}) \wedge (\neg x_{2,2} \vee \neg x_{5,2}) \wedge (\neg x_{2,3} \vee \neg x_{5,3}) \wedge$
$(\neg x_{3,1} \vee \neg x_{4,1}) \wedge (\neg x_{3,2} \vee \neg x_{4,2}) \wedge (\neg x_{3,3} \vee \neg x_{4,3}) \wedge$
$(\neg x_{3,1} \vee \neg x_{5,1}) \wedge (\neg x_{3,2} \vee \neg x_{5,2}) \wedge (\neg x_{3,3} \vee \neg x_{5,3})$

2.5 Hledání Hamiltonovské cesty pomocí SAT

Cílem této podkapitoly je převést problém hledání Hamiltonovské cesty na problém SAT. Úkol je definován následujícím způsobem: Máme dán (orientovaný) graf. Existuje v něm nějaká cesta, která obsahuje všechny uzly?

- Zakódování problému
  Předpokládejme, že graf G má U uzlů a H hran. Pro popis problému použijeme U^2 booleovských proměnných $x_{i,j}$, $1 <= i, j <= n$. Každá proměnná $x_{i,j}$ reprezentuje skutečnost, že uzel j je i-tým uzlem Hamiltonovské cesty.
- Každý uzel se musí být navštíven alespoň jednou:
  $(x_{1,j} \vee x_{2,j} \vee x_{3,j} \dots \vee x_{n,j})$ pro všechna j nabývající postupně hodnot $1 <= j <= n$
  Toto pravidlo vytvoří celkem n klauzulí.
- Žádný uzel nesmí být navštíven dvakrát:
  $(\neg x_{i,j} \vee \neg x_{k,j})$ pro všechna j nabývající postupně hodnot $1 <= j <= n, i != k$
  Toto pravidlo vytvoří celkem n*(n*(n-1)/2) klauzulí.
- Nějaký uzel musí být i-tý v pořadí:
  $(x_{i,1} \vee x_{i,2} \vee x_{i,3} \dots \vee x_{i,n})$
  Toto pravidlo vytvoří celkem n klauzulí.
- Žádné dva uzly nesmí být i-té v pořadí:
  $(\neg x_{i,j} \vee \neg x_{i,k})$ pro všechna i nabývající postupně hodnot $1 <= i <= n, j != k$
  Toto pravidlo vytvoří celkem n*(n*(n-1)/2) klauzulí.
- Poslední pravidlo se týká hran v grafu. Pokud hrana {i, j} není hranou v grafu, potom uzel j nemůže být následníkem uzlu i v Hamiltonovské cestě.
  $(\neg x_{k,i} \vee \neg x_{k+1,i})$ pro všechna k nabývající postupně hodnot $1 <= k <= n-1$
  Pro každé k dostáváme celkem n^2-H klauzulí, od nichž musíme odečíst (n-počet smyček) klauzulí, klauzule pro i=j jsou ignorovány, ale některé hrany mohou být

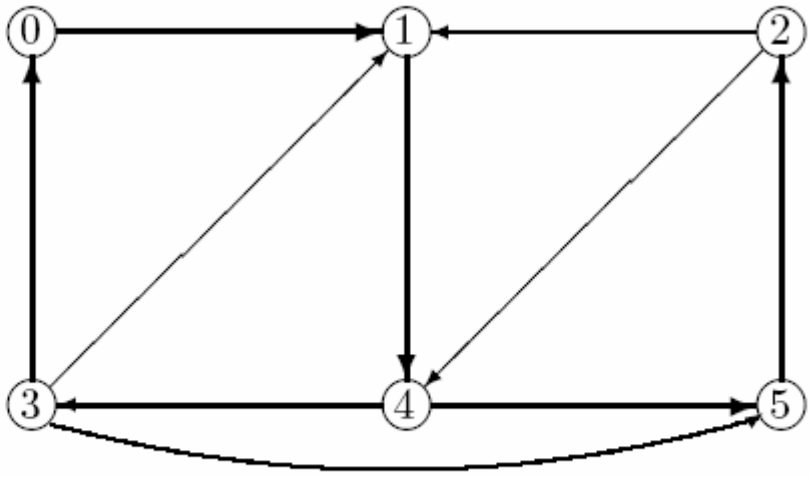
smyčky (u, u). Za cenu redundance můžeme prostě přidat hrany i=j, což dělá algoritmus jednodušší a zase tolik nevadí.

Celkový počet klauzulí vygenerovaný s využitím všech výše uvedených pravidel je následující:
2*n + 2*n*n*(n - 1)/2 + (n - 1)*(n^2 - H – n + počet smyček), což je zřejmě polynomiální složitost.

Výše uvedené zakódování generuje více klauzulí než je nutné. Např. v [Kro] se můžeme přesvědčit, že pravidla 2, 3 a 5 nebo 1, 4, 5 stačí k zakódování naší úlohy na SAT.

Celý převod si ještě ukážeme pro jednoduchý konkrétní příklad: graf s 6 uzly a 10 hranami.

```
p  6  10
e  0  1
e  1  4
e  2  1
e  2  4
e  3  0
e  3  1
e  3  5
e  4  3
e  4  5
e  5  2
```



Obr. 12 - Jednoduchý graf, v němž budeme hledat Hamiltonovskou cestu

Podle pravidla 1:
$(x_{1,0} \vee x_{2,0} \vee x_{3,0} \vee x_{4,0} \vee x_{5,0} \vee x_{6,0}) \wedge$
$(x_{1,1} \vee x_{2,1} \vee x_{3,1} \vee x_{4,1} \vee x_{5,1} \vee x_{6,1}) \wedge$
$(x_{1,2} \vee x_{2,2} \vee x_{3,2} \vee x_{4,2} \vee x_{5,2} \vee x_{6,2}) \wedge$
$(x_{1,3} \vee x_{2,3} \vee x_{3,3} \vee x_{4,3} \vee x_{5,3} \vee x_{6,3}) \wedge$
$(x_{1,4} \vee x_{2,4} \vee x_{3,4} \vee x_{4,4} \vee x_{5,4} \vee x_{6,4}) \wedge$
$(x_{1,5} \vee x_{2,5} \vee x_{3,5} \vee x_{4,5} \vee x_{5,5} \vee x_{6,5})$

Podle pravidla 2:
pro j=(1..n)
$(\neg x_{0,j} \vee \neg x_{1,j}) \wedge (\neg x_{0,j} \vee \neg x_{2,j}) \wedge (\neg x_{0,j} \vee \neg x_{3,j}) \wedge (\neg x_{0,j} \vee \neg x_{4,j}) \wedge (\neg x_{0,j} \vee \neg x_{5,j}) \wedge$
$(\neg x_{1,j} \vee \neg x_{2,j}) \wedge (\neg x_{1,j} \vee \neg x_{3,j}) \wedge (\neg x_{1,j} \vee \neg x_{4,j}) \wedge (\neg x_{1,j} \vee \neg x_{5,j}) \wedge$
$(\neg x_{2,j} \vee \neg x_{3,j}) \wedge (\neg x_{2,j} \vee \neg x_{4,j}) \wedge (\neg x_{2,j} \vee \neg x_{5,j}) \wedge$
$(\neg x_{3,j} \vee \neg x_{4,j}) \wedge (\neg x_{3,j} \vee \neg x_{5,j}) \wedge$
$(\neg x_{4,j} \vee \neg x_{5,j})$

Podle pravidla 3:

$(x_{0,1} \vee x_{0,2} \vee x_{0,3} \vee x_{0,4} \vee x_{0,5} \vee x_{0,6}) \wedge$
$(x_{1,1} \vee x_{1,2} \vee x_{1,3} \vee x_{1,4} \vee x_{1,5} \vee x_{1,6}) \wedge$
$(x_{2,1} \vee x_{2,2} \vee x_{2,3} \vee x_{2,4} \vee x_{2,5} \vee x_{2,6}) \wedge$
$(x_{3,1} \vee x_{3,2} \vee x_{3,3} \vee x_{3,4} \vee x_{3,5} \vee x_{3,6}) \wedge$
$(x_{4,1} \vee x_{4,2} \vee x_{4,3} \vee x_{4,4} \vee x_{4,5} \vee x_{4,6}) \wedge$
$(x_{5,1} \vee x_{5,2} \vee x_{5,3} \vee x_{5,4} \vee x_{5,5} \vee x_{5,6})$

Podle pravidla 4:
pro i=(1..n)
$(\neg x_{i,0} \vee \neg x_{i,1}) \wedge (\neg x_{i,0} \vee \neg x_{i,2}) \wedge (\neg x_{i,0} \vee \neg x_{i,3}) \wedge (\neg x_{i,0} \vee \neg x_{i,4}) \wedge (\neg x_{i,0} \vee \neg x_{i,5}) \wedge$
$(\neg x_{i,1} \vee \neg x_{i,2}) \wedge (\neg x_{i,1} \vee \neg x_{i,3}) \wedge (\neg x_{i,1} \vee \neg x_{i,4}) \wedge (\neg x_{i,1} \vee \neg x_{i,5}) \wedge$
$(\neg x_{i,2} \vee \neg x_{i,3}) \wedge (\neg x_{i,2} \vee \neg x_{i,4}) \wedge (\neg x_{i,2} \vee \neg x_{i,5}) \wedge$
$(\neg x_{i,3} \vee \neg x_{i,4}) \wedge (\neg x_{i,3} \vee \neg x_{i,5}) \wedge$
$(\neg x_{i,4} \vee \neg x_{i,5})$

Podle pravidla 5:
pro k=(1..n-1)
$(\neg x_{k,1} \vee \neg x_{k+1,0}) \wedge (\neg x_{k,0} \vee \neg x_{k+1,2}) \wedge (\neg x_{k,2} \vee \neg x_{k+1,0}) \wedge (\neg x_{k,0} \vee \neg x_{k+1,3}) \wedge$
$(\neg x_{k,0} \vee \neg x_{k+1,4}) \wedge (\neg x_{k,4} \vee \neg x_{k+1,0}) \wedge (\neg x_{k,0} \vee \neg x_{k+1,5}) \wedge (\neg x_{k,5} \vee \neg x_{k+1,0}) \wedge$
$(\neg x_{k,1} \vee \neg x_{k+1,2}) \wedge (\neg x_{k,1} \vee \neg x_{k+1,3}) \wedge (\neg x_{k,4} \vee \neg x_{k+1,1}) \wedge (\neg x_{k,1} \vee \neg x_{k+1,5}) \wedge$
$(\neg x_{k,5} \vee \neg x_{k+1,1}) \wedge (\neg x_{k,2} \vee \neg x_{k+1,3}) \wedge (\neg x_{k,3} \vee \neg x_{k+1,2}) \wedge (\neg x_{k,4} \vee \neg x_{k+1,2}) \wedge$
$(\neg x_{k,2} \vee \neg x_{k+1,5}) \wedge (\neg x_{k,3} \vee \neg x_{k+1,4}) \wedge (\neg x_{k,5} \vee \neg x_{k+1,3}) \wedge (\neg x_{k,5} \vee \neg x_{k+1,3})$

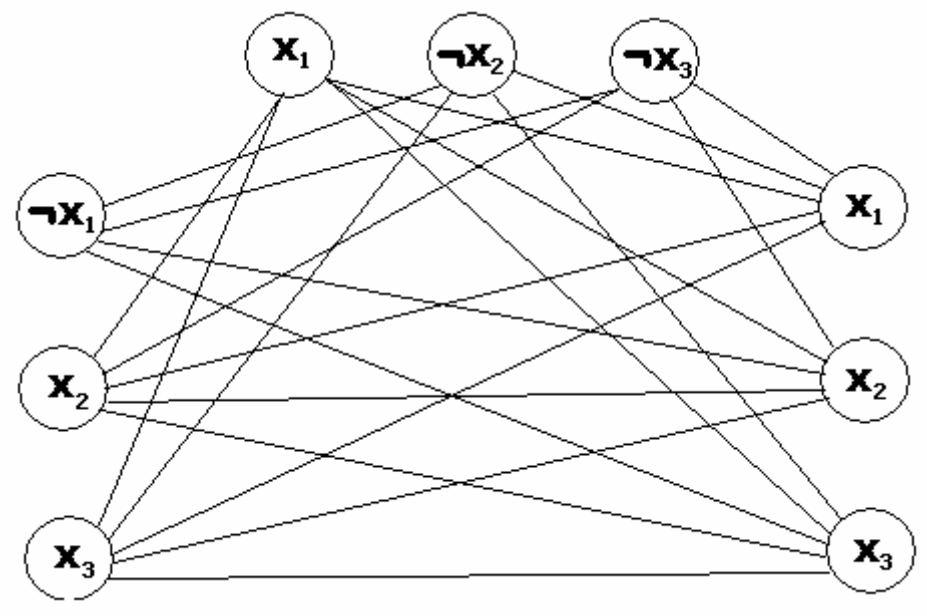
2.6 Převod mezi problémem SAT a hledáním kliky v grafu

Klika v grafu G je definována jako maximální úplný podgraf grafu G.

Mějme formuli $\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$. Provedeme transformaci na graf G podle následujících pravidel:

- každý literál každé klauzule představuje jeden uzel v grafu
- všechny uzly rozdělíme do tolika skupin, kolik klauzulí celkem máme
- mezi uzly provedeme spojení hranou v případě, kdy vrcholy nepatří do stejné skupiny a kdy jeden uzel není negací druhého

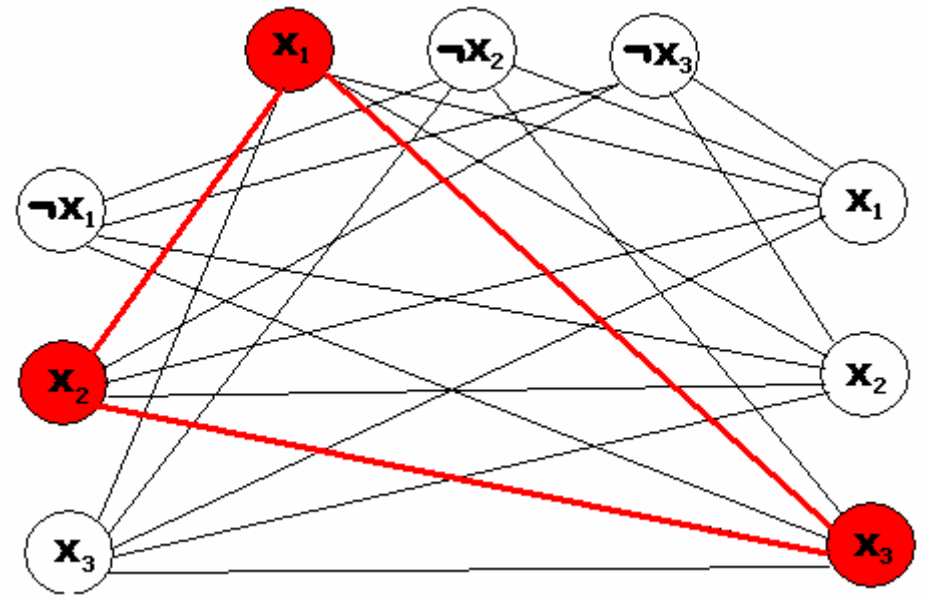
Pro formuli φ dostaneme následující graf:
3 klauzule způsobí rozdělení do tří skupin (vlevo, vpravo, nahoře), uzly ze stejné skupiny nejsou spojeny hranou, uzly z různých skupin označené proměnnými x_i a $\neg x_i$ nejsou spojeny hranou.



Obr. 13 - Formule zakódovaná do grafu

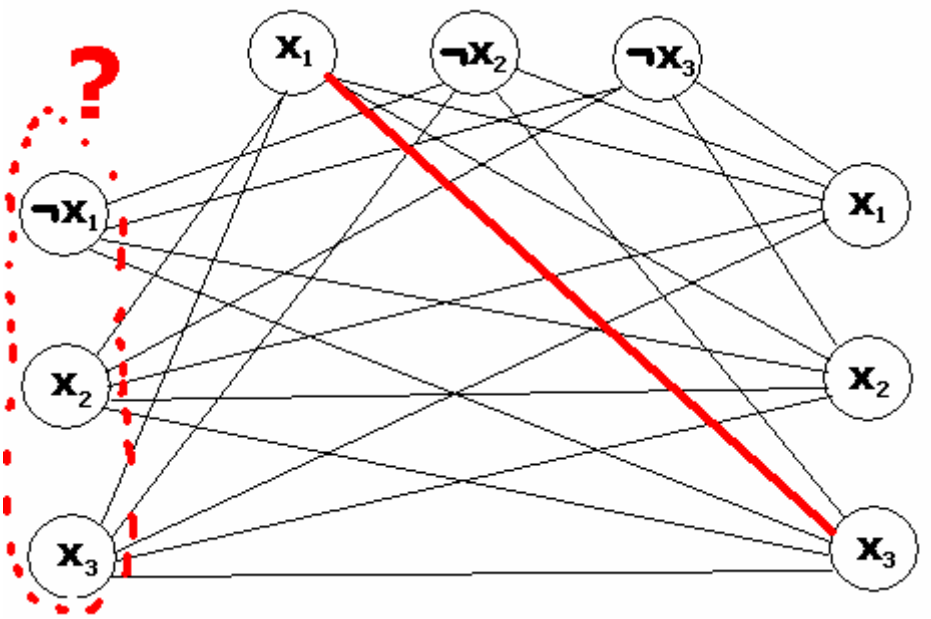
Ukažme, že platí: Pokud φ je splnitelná, pak G má kliku o velikosti rovné počtu klauzulí. Vzhledem k tomu, že se jedná o konjunkci klauzulí, v každé klauzuli musí být alespoň jeden literál, který má hodnotu 1. Kromě toho každému literálu ve φ odpovídá jeden vrchol v grafu G. Nakreslili jsme hrany, které byly konzistentní v tom smyslu, že vrcholy odpovídající literálům s hodnotou 1 jsou spojeny hranami s vrcholy z jiných skupin, které mají také hodnotu 1. To je zřejmě definice kliky.

Pro ohodnocení, ve kterém je formule splnitelná(x_1=1, x_2=1, x_3=1) dostaneme kliku o velikosti 3:



Obr. 14 - Ukázka, že ohodnocení, které splňuje formuli, vytváří kliku

Pro ohodnocení, ve kterém je formule nesplnitelná(x_1=1, x_2=0, x_3=1) nedostaneme kliku o velikosti 3 (literál x_2 v levé skupině v daném ohodnocení není roven 1, proto nebude „spojen“ a nebude tvořit kliku):



Obr. 15 - Ukázka, že ohodnocení, které nesplňuje formuli, nevytváří kliku

3 Obecně o genetických algoritmech

3.1 Co to je genetický algoritmus?

Jak si všichni jistě pamatujeme ze střední školy, genetika je věda, která se zabývá zkoumáním genetické informace živých organizmů. Každý jedinec je charakterizován svou genetickou informací, která určuje vlastnosti, chování a ovlivňuje významným způsobem život daného jedince. Tato informace je uložena v chromozómech.

Podle teorie evoluce přežívají v přírodě silnější jedinci a tito pak mají šanci předat svoji genetickou informaci do další generace. Důležité přitom je, že každý organismus je potomkem dvou rodičů, a tudíž se v něm mísí genetické informace obou rodičů. Genetická informace uložená v chromozómu potomka je tvořena částečně z genetické informace jednoho i druhého rodiče.

Na podobném principu pracuje i genetický algoritmus. Na počátku máme zadánu populaci náhodných jedinců, která se průběžně v souladu s teorií evoluce vyvíjí. Silnější jedinci přežívají, slabší jedinci hynou. Silnější jedinci mají větší šanci předat svou informaci dále svým potomkům, a tak dochází k zvýšení průměrné kvality celé populace. Tato vlastnost předurčuje algoritmus k použití na řešení optimalizačních problémů, tedy problémů, kdy hledáme nejlepší z možných řešení daného problému.

3.2 Struktura genetického algoritmu

Jak bylo popsáno v odstavci výše, každý jedinec má svou genetickou informaci. Velmi důležitá je reprezentace takové informace. Živý organizmus ji má uloženou v genech, které jsou dále uloženy v chromozómech, což jsou základní stavební prvky genetické informace.

V genetických algoritmech nemáme žádné geny, k reprezentaci genetické informace použijeme nějakou datovou strukturu, která bude co nejlépe vyhovovat řešení našeho problému. Tuto datovou strukturu můžeme chápat jako napodobení genu živého organizmu. Takovou datovou strukturou může být např. řetězec znaků 0, 1, kde číslo 1 může znamenat přítomnost a číslo 0 nepřítomnost daného prvku v konkrétní instanci.

Jak bylo popsáno v předchozím odstavci, genetický algoritmus využívá celé populace jedinců. Na počátku dojde vyplnění celé populace náhodně vygenerovanými jedinci. Následuje vždy křížení dvou rodičů a vzniku potomků. Výsledkem algoritmu je skutečnost, že ubývají méně kvalitní jedinci a přibývají více kvalitní jedince, a tedy průměrná kvalita jedince se zvyšuje. Jak ale algoritmus pozná, který jedince ze dvou je kvalitnější?

Zjištění kvality jedince neprovádí sám algoritmus, ale slouží k tomu hodnotící funkce (fitness), která vrací kvalitu daného jedince. Kvalita řetězce jako jediná přímo závisí na problému, který řešíme.

Dobu běhu genetického algoritmu můžeme rozdělit na několik fází:

- **Selekce**

Při selekci dochází ke kopírování řetězců ze staré generace do nové generace. Řetězec má tím větší šanci být zkopírován, čím kvalitnější je. Existují dvě základní metody, jak

provést výběr řetězců do nové generace, a to metodu vážené rulety(roulette wheel) a metodu turnaje(tournament selection).

Váženou ruletu si můžete představit jako klasickou ruletu pouze s jedním podstatným rozdílem - pravděpodobnost výběru je určena kvalitou řetězce.

Selekce metodou turnaje je velice jednoduchá metoda. Z generace vždy vybereme několik různých řetězců a do nové generace dáme silnější z nich. Je-li jedním z vybraných řetězců nejsilnější řetězec, je s pravděpodobností 1.0 propagován do další generace. Naopak nejhorší řetězec nemá žádnou šanci dostat se do další generace. Největší výhodou tohoto způsobu selekce je rychlost, s jakou k selekci dochází.

Můžeme pochopitelně zavést selekci založenou zcela na náhodě, tj. vybereme náhodného jedince z populace bez ohledu na jeho kvalitu a jiné vlastnosti.

Představme si následující problém. Máme populaci několika řetězců. Jeden řetězec je velmi silný, je velmi pravděpodobné, že je velmi blízko našemu hledanému řešení. Zbytek populace je zaplněn slabými a bezvýznamnými řetězci. V případě, že selekce proběhne přesně tak, jak bylo popsáno v tomto odstavci, může dojít k situaci, kdy tento jediný silný řetězec nebude vybrán a zanikne. Poté zbudou pouze slabé řetězce a informace obsažena v silném řetězci bude nenávratně ztracena. Abychom se takové situaci alespoň částečně vyhnuli, zavádíme další pojem – elitářství. Elitářství znamená, že n nejsilnějších řetězců z populace je zkopírováno do další generace bez toho, aby předtím muselo dojít ke křížení nebo k mutaci. Tím zabráníme nechtěnému ztracení informace v případě, že informačně nejsilnější řetězce nebudou vybrány.

- **Křížení**

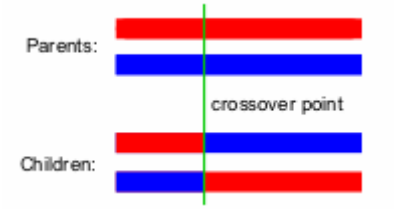
Křížení představuje výměnu informace mezi dvěma jedinci. Křížení probíhá následujícím způsobem. Z populace jsou vybrány dva řetězce – rodiče. Jejich potomci budou obsahovat část informace od každého rodiče. Následně je náhodně (nebo podle určitého pravidla) vybráno místo (nebo místa, pokud je způsob křížení jedinců složitější) v řetězci, kde dojde ke křížení. Poté následuje „roztržení“ genetických informací rodičů a vzniku dvou potomků. První potomek získá první část z prvního rodiče a druhou část genetické informace ze druhého rodiče, druhý potomek získá první část ze druhého rodiče a druhou část genetické informace z prvního rodiče. Nově vzniklé řetězce jsou vloženy do populace namísto původních rodičovských řetězců. Různé problémy mohou využívat různé způsoby křížení. Místo křížení může být stále stejné nebo náhodně proměnné, můžeme mít i více míst, kde dochází ke křížení. Křížení nemusí pouze kopírovat části řetězců jednotlivých rodičů, může zde docházet i ke změnám řetězce podle určitých pravidel. To vše však záleží na konkrétním algoritmu a jeho implementaci. Celý proces křížení je ukázán na následujícím jednoduchém příkladu:

rodič 1 1010010111
rodič 2 0101101010
místo dělení – mezi 2. a 3. bitem

10 10010111	=>	10 01101010	=>	1001101010
01 01101010		01 10010111		1010010111

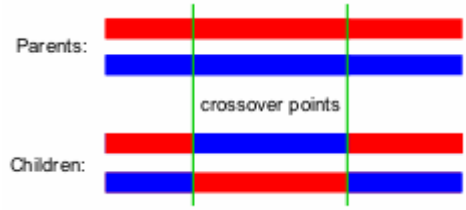
Dostali jsme tedy dva nové řetězce: 1001101010 a 1010010111.

Nakonec si uvedeme ještě přehled možných forem křížení, a to v přehledné grafické formě. Na počátku máme vždy dva rodiče, genetická informace jednoho rodiče je červeně znázorněná a druhého rodiče je modrá. Zelenou čarou označujeme místo nebo místa, kde dochází ke křížení.



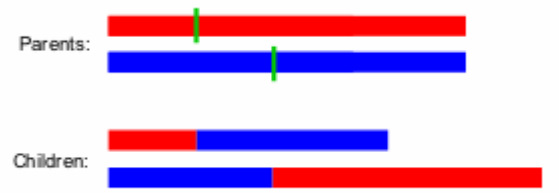
Obr. 16 - Názorná ukázka křížení metodou Simple crossover

V případě jednobodového křížení dojde k prohození genetické informace obou rodičů za místem křížení.



Obr. 17 - Názorná ukázka křížení metodou Two-point crossover

V případě dvoubodového křížení dojde k prohození genetické informace obou rodičů mezi oběma místy křížení.



Obr. 18 - Názorná ukázka křížení metodou „Cut and splice“ crossover

V případě křížení „cut and splice“ dojde k prohození genetické informace obou rodičů za místem křížení. Tato metoda se liší od dříve uvedené metody tím, že místo, kde dochází ke křížení, není u obou rodičů stejné. Vznikají tak potomci, jejichž délka se může lišit od původní délky rodičů. Uvědomme si, že tato metoda křížení nemusí být vhodná pro některé úlohy, konkrétně pro SAT vhodná není, neboť nezachovává délku řetězců.

Poslední formou křížení, kterou si představíme, je „uniform crossover“. V tomto případě jsou znaky řetězců jednotlivých rodičů prohozeny typicky s pevně danou pravděpodobností 0,5. Je třeba si uvědomit, že prohazujeme vždy n-tý znak jednoho rodiče s n-tým znakem druhého rodiče.
Můžeme jistě vymyslet ještě mnoho dalších forem křížení, které se mohou více či méně hodit pro řešení určité specifické úlohy. Tento odstavec si pouze kladl za cíl seznámit čtenáře s nejčastěji využívanými možnostmi. Dále viz. např. [Wik].

- **Mutace**

Stejně jako v přírodě může dojít v naší populaci řetězců k mutaci – náhodné změně určitého „genu“. Mutace je realizována následujícím způsobem: Procházíme postupně populaci našich řetězců, bereme jednoho jedince za druhým a na každý znak řetězce

charakterizujícího daný řetězec zavoláme s určitou pravděpodobností metodu měnící náhodně hodnotu daného paměťového místa. Celý proces mutace je opět ukázán na jednoduchém příkladu:

Původní řetězec: 1010010111
dojde k mutaci na třetím bitu

Zmutovaný řetězec: 1000010111

Oba řetězce (původní i zmutovaný) jsou zcela totožné s výjimkou místa, kde došlo k mutaci.

4 Problém SAT a genetické algoritmy

4.1 Úvod do problematiky

V této kapitole se pokusíme využít výhody genetických algoritmů k řešení problému splnitelnosti booleovské formule. Navrhneme vhodnou implementaci vnitřní formy a prodiskutujeme a zhodnotíme možné hodnotící funkce fitness.

4.2 Vnitřní reprezentace

Abychom mohli aplikovat metodu genetických algoritmů na konkrétní problém, musíme zvolit nějakou vhodnou vnitřní reprezentaci řetězce. V případě problému SAT je velmi vhodné zvolit tuto řetězcovou reprezentaci. Formuli o n proměnných zakódujeme do řetězce o konstantní délce n, kde i-tý bit řetězce reprezentuje pravdivostní hodnotu i-té booleovské proměnné ze všech n proměnných přítomných v zadané formuli. Každá proměnná bude moci nabývat dvou základních hodnot (0 ... false, 1 ... true).

Poznámka: Je třeba si uvědomit, že pojem „bit“ nemusí vždy označovat pouze jednotku informace o dvou hodnotách. V tomto případě je myšleno pod pojmem bit paměťové místo v reprezentaci řetězce uchovávající informaci o hodnotě nejmenší datové jednotky v daném algoritmu (jednoho znaku řetězce). Jeden znak řetězce může (ale také nemusí) zabírat v paměti místo o velikosti 1 bit.

Např.:
formule $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$ obsahuje tři proměnné x_1, x_2, x_3 a zakódujeme ji tedy řetězcem o délce 3 znaky.
Genetická informace jedince patřícího do populace na vyřešení výše uvedeného problému může vypadat např. takto: 000, 001, 010, 011, 100, 101, 110, 111

Je těžké představit si lepší reprezentaci problému vhodnou pro řešení pomocí genetických algoritmů: tato reprezentace má pevnou délku řetězce, je „kontextově nezávislá“, tj. význam jednoho bitu se nemění změnou hodnot ostatních bitů.

4.3 Výběr hodnotící (fitness) funkce

Trochu více přemýšlení vyžaduje výběr hodnotící funkce. Nejjednodušší a nejintuitivnější řešení je zavedení takové funkce, která vrátí hodnotu 1, pokud řešení reprezentované řetězcem je splnitelné (vrací hodnotu TRUE), nebo hodnotu 0 v opačném případě. Pokud se trochu zamyslíme nad tímto prvním návrhem možné podoby hodnotící funkce, dojdeme k závěru, že hodnotící funkcí navracená hodnota bude téměř ve všech případech rovna 0 a její výsledky nám moc nepomohou v případě selekce. Není problém si představit populaci řetězců, které všechny budou nesplněné(budou vracet hodnotu 0). Jak potom vybereme řetězce vhodné pro křížení a selekci?

Jako další návrh funkce fitness uvedeme funkci φ takovou, která vrací celkový počet splněných (resp. nesplněných) klauzulí v řešené formuli. Pro výraz skládající se z n klauzulí

vrací funkce φ hodnoty celých čísel v intervalu 0 až n. V případě navracení počtu nesplněných klauzulí je naším cílem vygenerovat takový řetězec, pro který platí, že na něj zavolaná hodnotící funkce vrátí hodnotu 0.

Uveďme si jednoduchý příklad:
Máme formuli $\varphi = (x_1 \vee x_3 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_5 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee x_4)$ a ohodnocení proměnných 11001 (x_1= 1, x_2= 1, x_3= 0, x_4= 0, x_5= 1).
clausesSat = $\{(1 \vee 0 \vee 1) \wedge (\neg 1 \vee 0 \vee \neg 1) \wedge (\neg 1 \vee \neg 1 \vee 0) \wedge (1 \vee \neg 1 \vee 0) \wedge (1 \vee 0 \vee 0) \wedge (\neg 0 \vee \neg 0 \vee 1) \wedge (\neg 1 \vee 0 \vee 0)\} = \{1 \wedge 0 \wedge 0 \wedge 1 \wedge 1 \wedge 1 \wedge 0\} = 4$.
Analogicky lze určit velikost hodnoty clausesUnsat(počet nesplněných klauzulí)
clausesUnsat = numberOfClauses – clausesSat = 7 – 4 = 3.

Zkusme provést ještě následující variantu:
$value(x_1 \wedge x_2 \wedge x_3 \ldots \wedge x_n) = AVE\ (value(x_1), value(x_2), value(x_3), \ldots value(x_n))$,
kde AVE označuje průměrnou hodnotu (average)

Tento návrh byl původně navržen Smithem (*Adaptive Genetic Algorithms and the Boolean Satisfiability Problem*). Jeho výhodou je to, že kromě hodnot „pravda“ a „nepravda“ vrací také hodnoty „je více pravda“ a „je více nepravda“. Ukažme si výhodu tohoto návrhu na následujícím příkladě:

$x_1 \wedge (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

x_1	x_2	x_3	fitness
0	0	0	1/3
0	0	1	1/3
0	1	0	2/3
0	1	1	1/3
1	0	0	3/3
1	0	1	3/3
1	1	0	3/3
1	1	1	3/3

Tab. 1 – Výpočet fitness funkce pomocí metody AVE

Je zřejmé, že ohodnocení 010 je „lepší než „011“, neboť f(010) = 2/3 > f(011) = 1/3.
Je dobré si všimnout, že ve všech případech, kdy je formule splněná, je hodnota fitness rovna 3/3 = 1. Toto řešení je výhodné v situaci, kdy nemají všechny klauzule stejný počet literálů.

Poslední variantou, kterou si v této kapitole představíme, je vážený MAX-SAT. Máme zadánu funkci F o *n* proměnných $X = \{x_1, x_2 \ldots x_n\}$. Každá proměnná má vlastní váhu $W = \{ w_1, w_2 \ldots w_n \}$ Naší snahou je nalézt takové řešení, kdy je klauzule splněná a součet vah proměnných ohodnocených 1 je maximální. Od tohoto součtu vah může být odečten počet nesplněných klauzulí zatížených(vynásobených) určitou penalizací. Jako příklad si uvedeme takovou funkci fitness, která vrací hodnotu h = součet vah všech jedniček -1000*počet nesplněných klauzulí.

Vše vysvětluje následující příklad:
n = 4
F = (x1 + ¬x3 + x4).(¬x1 + x2 + ¬x3).(x3 + x4).(x1 + x2 + ¬x3 + ¬x4).

(¬x2 + x3).(¬x3 + ¬x4)
W = (2, 4, 1, 6)

Možná řešení, kdy F = 1:
X = {x1 ... xn} = {0, 0, 0, 1}, S = 6
X = {x1 ... xn} = {1, 0, 0, 1}, S = 2 + 6 = 8
X = {x1 ... xn} = {1, 1, 1, 0}, S = 2 + 4 + 1 = 7

=> Vybereme druhé řešení se součtem 8.

4.4 Operátor křížení

V kapitole 3 jsme popsali tři základní varianty operátoru křížení, a to křížení jednobodové, dvoubodové a uniformní. V případě řešení problému SAT byl implementován ještě další způsob křížení, a to v souladu s návrhem uvedeným např. v [Hao]. Nazveme jej „zlepšující křížení“ (crossover improvement).

Definice nového typu křížení je následující: vybereme dva jedince X a Y některou z metod uvedených v kapitole 3. Pro každou klauzuli takovou, že je nesplněná v obou ohodnoceních X a Y. Pro všechny proměnné v nesplněných klauzulích vypočítáme hodnotu σ_i = X.imp(i) + Y.imp(i) a nastavíme Z(k) = flip(X(k)) na místě, kde σ_k je maximální. Pokud X(k) = Y(k), pak Z(k) = X(k). V případě, že X(k) je různé od Y(k), nastavíme hodnotu Z(k) na náhodnou hodnotu.

Popišme ještě funkci φ takovou, která vrací „zlepšení“ získané překlopením jedné proměnné. Pod pojmem zlepšení si můžeme představit číslo, které určuje počet nesplněných klauzulí, které se po překlopení proměnné staly splněnými mínus počet splněných klauzulí, které se po překlopení proměnné staly nesplnitelnými. K překlopení vybereme tu proměnnou, jejíž hodnota fitness je nejvyšší.
Uveďme si jednoduchý příklad:
Máme formuli φ stejnou jako v předchozím případě, ohodnocení proměnných je také stejné. Určíme hodnotu imp(2), tj. zlepšení hodnoty s indexem 2.
Původní ohodnocení: 11001 (x_1= 1, x_2= 1, x_3= 0, x_4= 0, x_5= 1).
clausesSat = 4 (stejně jako v předchozím případě)
Nové ohodnocení: 10001 (x_1= 1, x_2= 0, x_3= 0, x_4= 0, x_5= 1).
clausesSat = {(1 ∨ 0 ∨ 1) ∧ (¬0 ∨ 0 ∨ ¬1) ∧ (¬1 ∨ ¬0 ∨ 0) ∧ (1 ∨ ¬1 ∨ 0) ∧ (0 ∨ 0 ∨ 0) ∧
(¬0 ∨ ¬0 ∨ 1) ∧ (¬0 ∨ 0 ∨ 0)} = {1 ∧ 1 ∧ 1 ∧ 1 ∧ 0 ∧ 1 ∧ 1} = 6
Výsledek je tedy imp(2) = 6 – 4 = 2.

Následující jednoduchý příklad ilustruje tento typ křížení:
Máme formuli φ = (x_1 ∨ x_3 ∨ x_5) ∧ (¬x_2 ∨ x_3 ∨ ¬x_5) ∧ (¬x_1 ∨ ¬x_2 ∨ x_4) ∧ (x_1 ∨ ¬x_5 ∨ x_4) ∧ (x_2 ∨ x_3 ∨ x_4) ∧ (¬x_3 ∨ ¬x_4 ∨ x_5) ∧ (¬x_2 ∨ x_3 ∨ x_4) a ohodnocení proměnných X = 11001 (x_1= 1, x_2= 1, x_3= 0, x_4= 0, x_5= 1) a Y = 01001 (x_1= 0, x_2= 1, x_3= 0, x_4= 0, x_5= 1).
Zjistíme, které formule jsou nesplněné jak v ohodnocení X, tak v ohodnocení Y:
o_x = (1 ∨ 0 ∨ 1) ∧ (¬1 ∨ 0 ∨ ¬1) ∧ (¬1 ∨ ¬1 ∨ 0) ∧ (1 ∨ ¬1 ∨ 0) ∧ (1 ∨ 0 ∨ 0) ∧ (¬0 ∨ ¬0 ∨ 1) ∧
(¬1 ∨ 0 ∨ 0) = 1 ∧ 0 ∧ 0 ∧ 1 ∧ 1 ∧ 1 ∧ 0
o_y = (0 ∨ 0 ∨ 1) ∧ (¬1 ∨ 0 ∨ ¬1) ∧ (¬0 ∨ ¬1 ∨ 0) ∧ (0 ∨ ¬1 ∨ 0) ∧ (1 ∨ 0 ∨ 0) ∧ (¬0 ∨ ¬0 ∨ 1) ∧
(¬1 ∨ 0 ∨ 0) = 1 ∧ 0 ∧ 1 ∧ 0 ∧ 1 ∧ 1 ∧ 0
Z výpočtů výše je vidět, že v obou ohodnoceních jsou nesplněné klauzule (¬x_2 ∨ x_3 ∨ ¬x_5) a (¬x_2 ∨ x_3 ∨ x_4).

Vypočtěme nyní zlepšení postupně všech proměnných z obou klauzulí, které vznikne překlopením jedné proměnné.

překlopená proměnná	ohodnocení X	splněno	ohodnocení Y	splněno	zlepšení X	zlepšení Y	celkové zlepšení
-	11001	4	01001	4	-	-	-
x_2	10001	6	00001	5	6 – 4 = 2	5 – 4 = 1	2 + 1 = 3
x_3	11101	6	01101	6	6 – 4 = 2	6 – 4 = 2	2 + 2 = 4
x_4	11011	6	01011	6	6 – 4 = 2	6 – 4 = 2	2 + 2 = 4
x_5	11000	5	01000	5	5 – 4 = 1	5 – 4 = 1	1 + 1 = 2

Tab. 2 – Ukázka metody křížení improvement (zlepšení)

Z výše uvedené tabulky je vidět, že největšího zlepšení dosáhneme invertováním proměnných x_3 nebo x_4. Obě proměnné mají stejné zlepšení a podle pravidel ze začátku této kapitoly budeme obě tyto proměnné přenášet do následníka v invertované podobě, tedy Z = xx11x (x značí zatím neurčenou hodnotu proměnné).
Proměnné, které jsou ohodnoceny stejně v ohodnocení X a Y (a nedošlo k jejich změně v předchozím kroku), budou přeneseny do ohodnocení Z beze změny, tj. druhá a sedmá proměnná v ohodnoceních X a Y jsou v obou případech rovny jedné, tj. Z = x1111.
Zbývá nám poslední proměnná, jejíž hodnota nebyla určena žádnou z předchozích metod. Přiřadíme tedy této proměnné hodnotu náhodnou, např. 0. Ze dvou rodičů X a Y jsme vytvořili jednoho následníka Z
Jak si ale můžeme všimnout v kapitole 2, každý operátor křížení dostane na vstupu dva rodiče a na výstupu vygeneruje dva potomky, kdežto náš nový operátor zlepšení vygeneruje ze dvou rodičů jednoho potomka. Tento problém můžeme vyřešit několika možnými způsoby:
1) Vytvoříme dva nové potomky, které se budou lišit náhodně přiřazenými proměnnými. V předchozím případě jsme si ukázali příklad, kdy z rodičů X a Y vznikl potomek Z. Ohodnocení proměnných potomka Z bylo jednoznačně určeno pravidly z předchozího odstavce s výjimkou proměnné x_1, jejíž hodnotu jsme přiřazovali náhodně. Je tedy možné vytvořit dva následníky Z a Z' takové, že náhodně přiřazovaná proměnná bude v prvním případě rovna jedné a druhém případě rovna nule. Dostaneme tak Z = 01111 a Z' = 11111.
2) Jedním následníkem bude Z, druhým následníkem bude ten rodič z dvojice X, Y, jehož fitness bude vyšší. Znamená to tedy, že jeden rodič bude distribuován do nové generace beze změny.

4.5 Formát DIMACS

Všechny profesionální nástroje a softwarové produkty řešící problém SAT musí před tím, než začnou počítat výsledek, získat (načíst) informace o struktuře formule, počtu proměnných, podobě jednotlivých klauzulí apod. Získání těchto informací z formulí zapsaných pomocí stejných symbolů tak, jako známe z matematické logiky, by bylo složité a neefektivní. Aby bylo usnadněno přenášet stejná zadání mezi různými programy a tím umožněno testovat výkonnost jednotlivých algoritmů a metod, byl navržen speciální formát DIMACS. K nejdůležitějším patří dva níže uvedené formáty.

1) Formát CNF

Toto je jednoduché schéma formátu:
c comment
p cnf numVars numClauses
list of clauses

- písmeno c na začátku řádku značí komentář (comment)
- písmeno p na začátku řádku značí seznam parametrů (parameter). Hned za označením řádku se nachází forma zadané formule (např. cnf), kterou následují další dvě hodnoty, a to počet proměnných celkem a počet klauzulí celkem.
- následuje už jen seznam všech klauzulí. Každé číslo znamená index proměnné, znaménko mínus označuje negaci. Každý řádek je navíc zakončen nulou.

příklad:
//začátek souboru tst.cnf
c Optional comments
c at start of file.
c The "p" line specifies cnf format, number vars, number clauses
p cnf 3 2
1 -3 2 0
2 3 -1 0
//konec souboru tst.cnf

Soubor tst.cnf reprezentuje zřejmě tuto formuli:

$(x_1 \vee \neg x_3 \vee x_2) \wedge (x_2 \vee x_3 \vee \neg x_1)$

Formule má 3 různé proměnné x_1, x_2, x_3 a skládá se ze dvou klauzulí.

2) Formát SAT

Toto je jednoduché schéma formátu:
c comment
p sat numVars
list of clauses

- písmeno c na začátku řádku značí komentář (comment)
- písmeno p na začátku řádku značí seznam parametrů (parameter). Hned za označením řádku se nachází forma zadané formule (např. sat), kterou následuje hodnota určující počet proměnných celkem
- následuje už jen seznam všech klauzulí. Každé číslo znamená index proměnné, znaménko mínus označuje negaci.
- operátor „+“ označuje operaci OR, operátor „*“ označuje operaci AND, operátor „-“ označuje operaci NOT

příklad:
//začátek souboru tst.sat
c Optional comments
c at start of file.
c The "p" line specifies cnf format, number vars, number clauses
p sat 4

```
(*(     +(1 3 -4)
        +(4)
        +(2 3)))
//konec souboru tst.sat
```

Soubor tst.sat reprezentuje zřejmě tuto formuli:

$(x_1 \vee x_3 \vee \neg x_4) \wedge (x_4) \wedge (x_2 \vee \neg x_3)$

Na závěr je třeba dodat, že formát CNF je běžnější a více podporován. Další informace k formátům SAT a CNF jsou např. v [Ssf].

4.5 Vlastní implementace - program SatGen

SatGen je program řešící problém SAT pomocí genetických algoritmů. Celé řešení jsem naprogramoval od základu zcela sám v prostředí CBuilderX. Celý systém se skládá z několika komponent.

- parser - načtení formátu klauzule ze souboru
- CString - reprezentace vnitřní formy uložené klauzule, definice fitness funkcí
- CGeneration - vlastní genetický algoritmus
- form1, paintpane - GUI
- interface - nastavení druhu kompilace

Podrobnosti k výše uvedeným souborům a třídám viz. Příloha P.4.

Předtím, než začne program vyhledávat řešení, je nutné, aby uživatel zadal několik parametrů genetického algoritmu. Jedná se zejména o tyto parametry: počet jedinců v populaci, počet elitních jedinců, počet generací (jak dlouho se má nechat populace vyvíjet), pravděpodobnost křížení, pravděpodobnost mutace, metody selekce, křížení, výběr fitness funkce...

Není však jednoduché říct, jak je vhodné tyto parametry nastavit. Např. s rostoucím počtem jedinců v generaci dochází k zvětšení prohledávaného prostoru možných řešení, ale tím dochází také ke zvětšení nároků na paměť a čas, neboť počítáme najednou více takových hodnot.

Snadná nastavitelnost programu umožňuje uživateli experimentovat se zadanými parametry a dosahovat tím co nejlepších možných výsledků.

Na obrázcích uvedených v kapitolách 5.1.1 a 5.1.2 vidíme příklad vývoje populace. V levé části okna vidíme všechny parametry populace, v pravé části okna vidíme průběh vývoje populace. Zeleně je označen vývoj nejlepšího dosud nalezeného řešení, modře je zobrazen vývoj průměrného jedince populace, červeně je vyjádřen vývoj nejhoršího jedince.

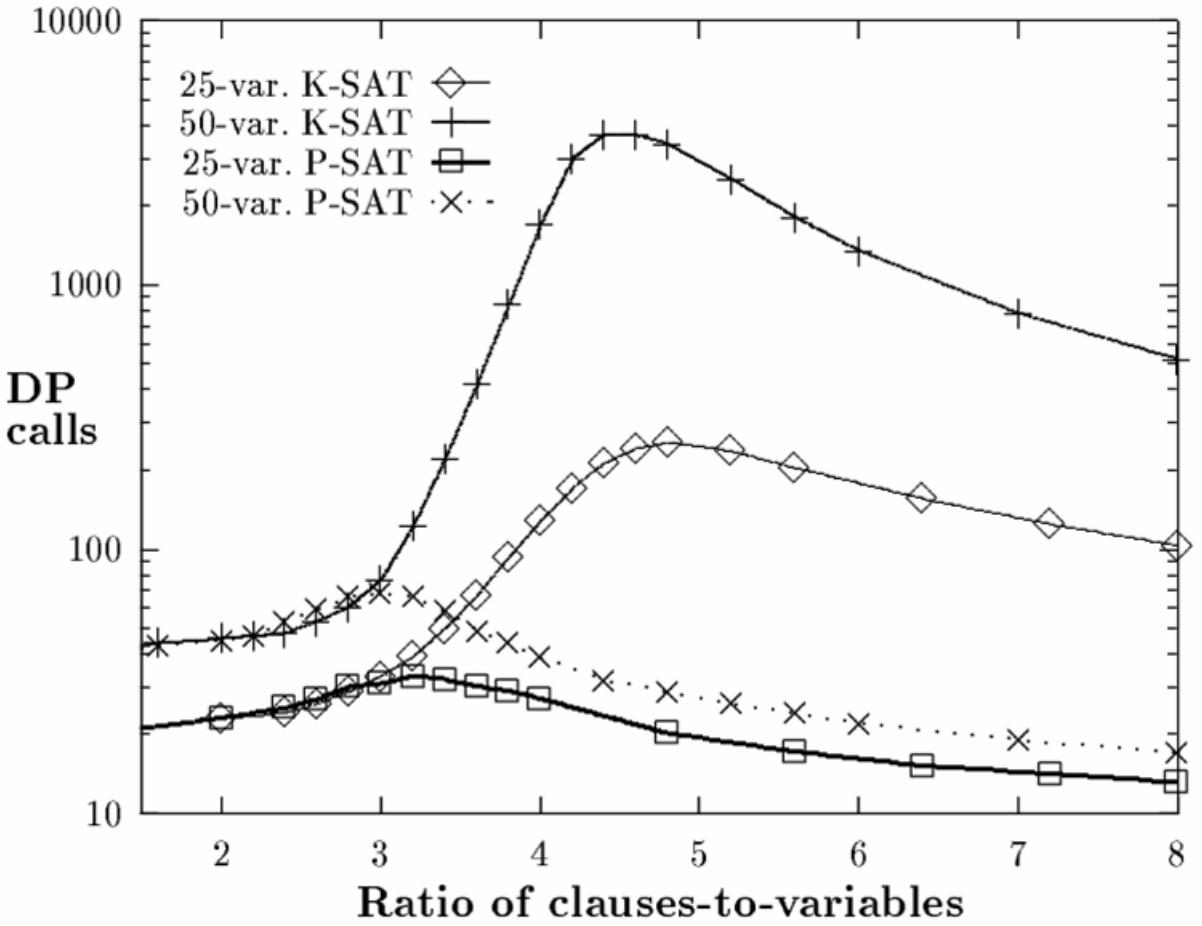
4.6 Vlastní implementace - program randSat

Program randSat slouží ke generování náhodných booleovských formulí ve formátu DIMACS 3-CNF. Všechny testovací soubory a formule jsou vytvořeny s pomocí tohoto programu. Generování booleovských formulí je zcela založeno na náhodě. Do každé vytvářené klauzule vybereme tři různá čísla reprezentující proměnnou a tuto proměnnou navíc ještě s pravděpodobností 0,5 znegujeme. Program nám umožňuje zadávat různý poměr generovaných klauzulí a proměnných.

Nyní se nabízí otázka, jak vhodně zvolit tento poměr. Je jasné, že pokud zvolíme např. jednu proměnnou a deset klauzulí, tak bude formule s velkou pravděpodobností nesplnitelná(všechny klauzule určitě nebudou úplně stejné, abychom následným dosazením hodnoty jedné proměnné dokázali všechny klauzule splnit). Opačný extrém ilustruje situace, kdy zvolíme např. deset proměnných a deset klauzulí. Taková klauzule bude naopak s velkou pravděpodobností splnitelná, jak se může přesvědčit spočítáním několika málo jednoduchých příkladů. Zkusme se zamyslet nad velikostí tohoto poměru.

Definujme metodu (v literatuře označovanou jako DP), která prohledává do hloubky stavový prostor obsahující všechna možná přiřazení hodnot proměnným.
Metoda DP:

- dána množina klauzulí M definovaná s využitím množiny proměnných V
- nastav hodnotu proměnné v a zavolej DP na takto změněné formuli, pokud tato formule je splněna, ukonči prohledávání
- invertuj proměnnou v a vrať výsledek DP volané na takto změněné přiřazení proměnných



Obr. 19 - Závislost počtu volání metody DP na poměru c = počet klauzulí/počet proměnných

Je-li *c*>4.762, náhodně vygenerovaná 3-SAT formule je nesplnitelná s vysokou pravděpodobností [Kamath et al '94]. Více informací k tomuto problému viz. např. [Sel]. Jak je vidět na výše uvedeném grafu, nejsložitější je vyřešit takové formule, které mají poměr klauzule:proměnné v intervalu 4 až 6. Mysleme proto na tyto skutečnosti při generování formulí, které chceme řešit.

Program spustíme z příkazové řádky s následujícími parametry.

randSat [jméno výstupního souboru] [počet proměnných] [počet klauzulí]

Příklad vstupu: **randSat tst_v10_c100.cnf 10 100** vytvoří sto klauzulí, deset proměnných a vše uloží do souboru tst_v10_c100.cnf.

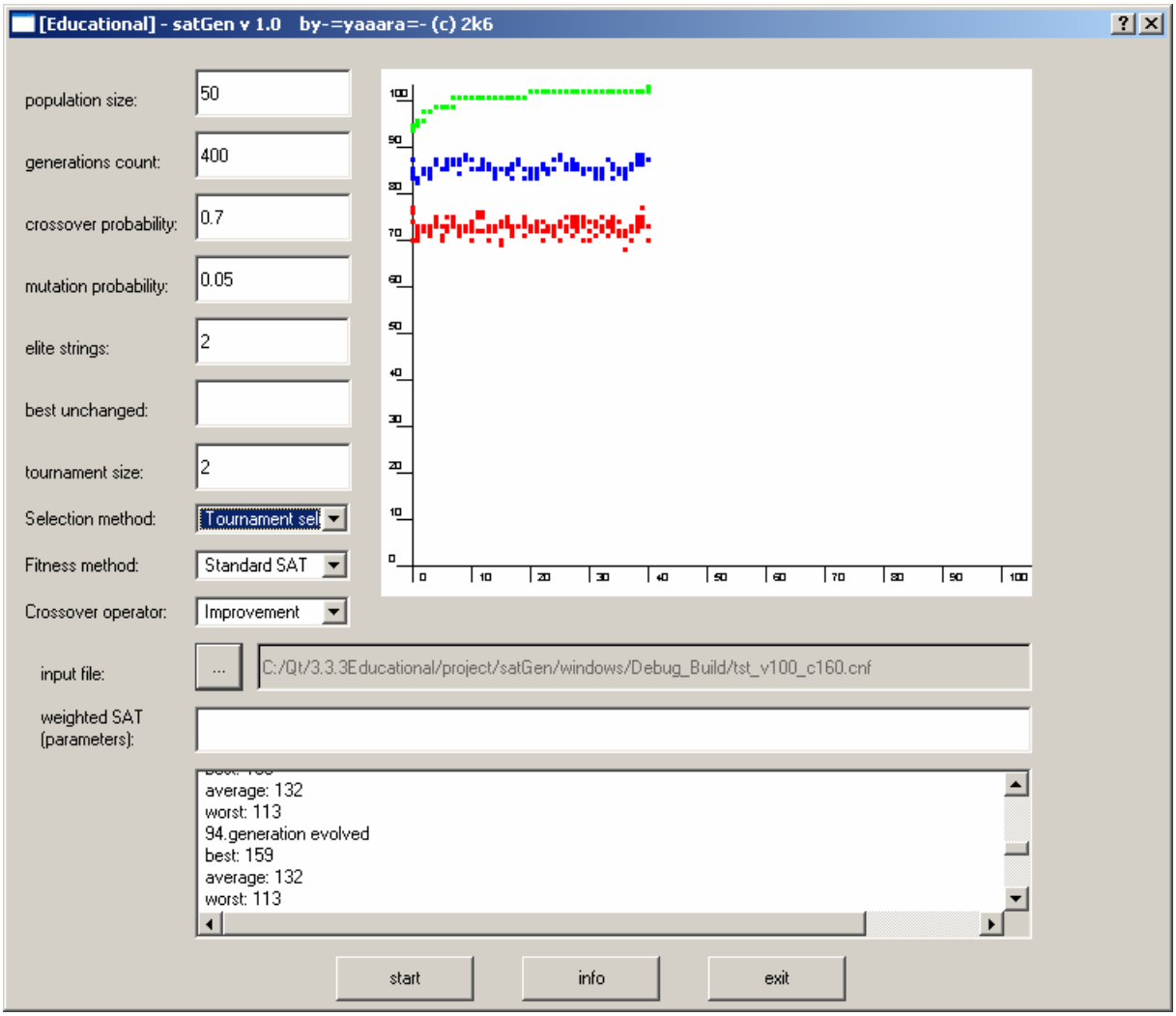
Kompletní zdrojový kód je včetně komentářů uveden v příloze této zprávy.

5 Experimenty

5.1 Porovnání standardního a váženého SATu

5.1.1 Standardní SAT

Vzhledem k tomu, že před dosažením maximálního počtu generací(zde 400) došlo k nalezení řešení, tak „graf“ nekončí na konci zobrazovací plochy. Pro úplnost je dobré zmínit, že svislá a vodorovná osa udává hodnotu v procentech, tj. vodorovná osa říká, kolik procent z „generations count“ uběhlo, a svislá osa udává, kolik procent klauzulí z celku je splněných.

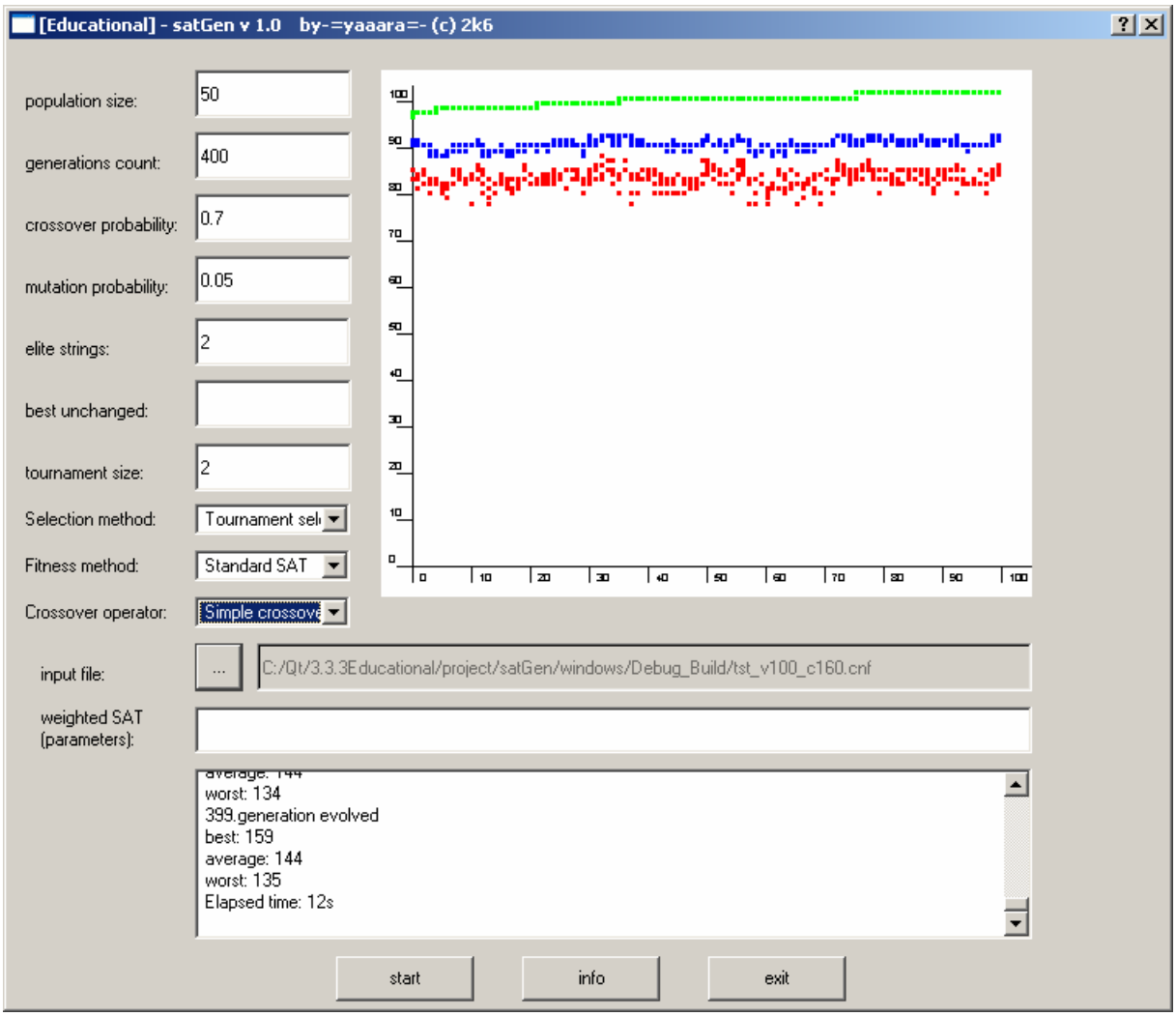


Obr. 20 - Ukázka grafického výstupu – standard SAT/tournament2/improvement

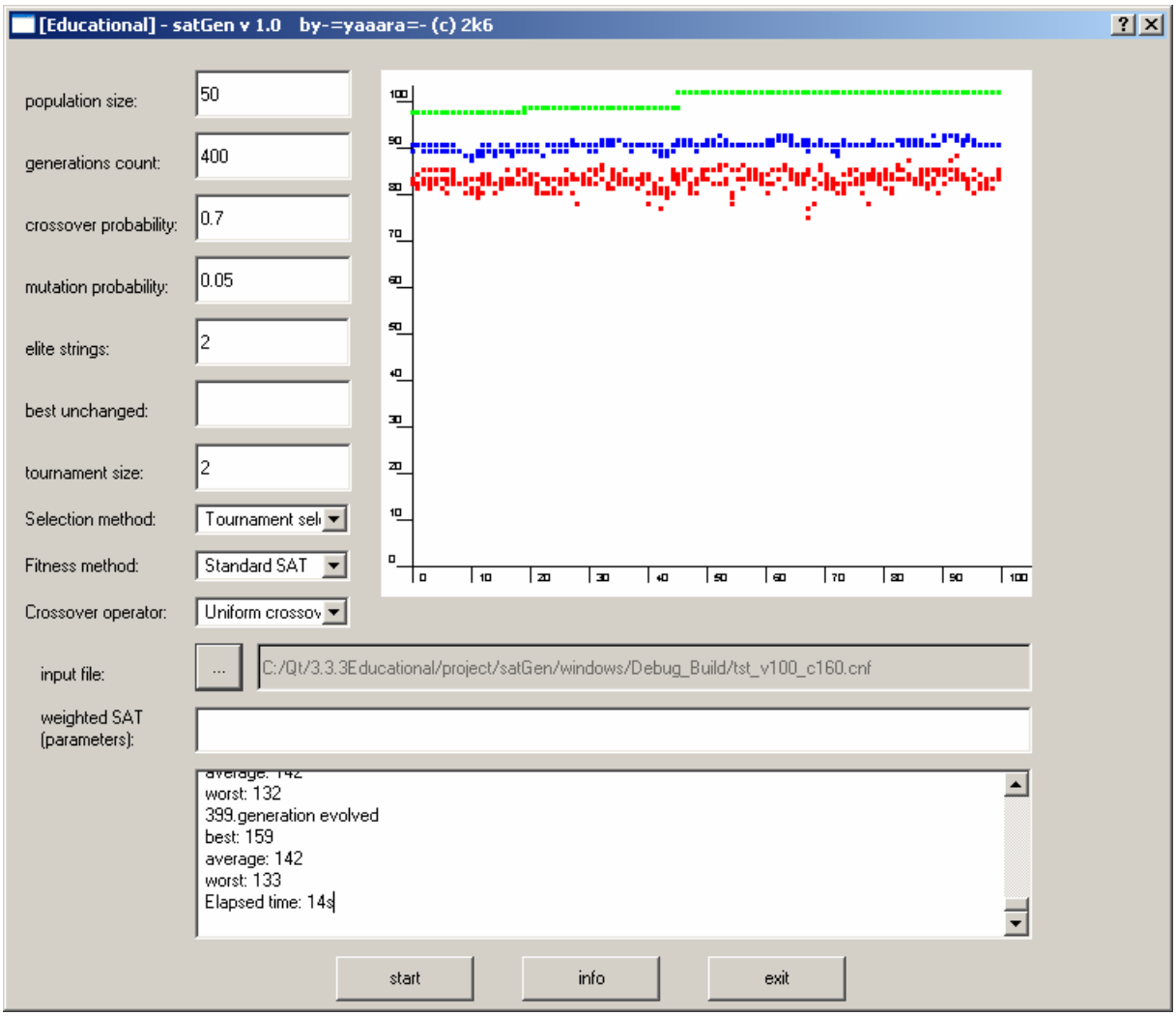
Zkusme porovnat průběhy na obrázcích výše a níže. Na obrázku výše je ukázána metoda tournament2/improvement, na obrázku níže je ukázána metoda tournament2/simple. Zde jsou vidět některé přednosti metody improvement:

1) řešení je nalezeno dříve(v žádném ze dvou dalších příkladů není řešení nalezeno) Skutečnost, zda řešení bylo nalezeno, poznáme z textového pole ve spodní okna aplikace – pokud se zde objeví nápis „Solution:”, tak vidíme, že řešení bylo nalezeno. Informaci o tom, zda bylo nalezeno řešení, můžeme zjistit jednodušeji na první pohled ze zobrazeného grafu. Graf na výše uvedeném obrázku „končí” zhruba uprostřed rozsahu vodorovné osy – bylo již nalezeno řešení a nemá smysl hledat dál. Naopak na níže uvedeném obrázku prochází graf v celém rozsahu vodorovné osy, takže řešení pravděpodobně nalezeno nebylo.

2) hodnota nalezeného nejlepšího řešení se „stále mění”, kdežto hodnota nejlepšího nalezeného řešení u metody tournament2/simple se mění „méně častěji” a nejlepší hodnota v případě tournament2/uniform se mění pouze po začátku běhu algoritmu, a pak zůstává víceméně konstantní



Obr. 21 - Ukázka grafického výstupu – standard SAT/tournament2/simple



Obr. 22 - Ukázka grafického výstupu – standard SAT/tournament2/uniform

5.1.2 Vážený SAT

Ještě si ukažme, jak bude výpočet probíhat v případě použití váženého SATu(viz. níže). Parametry pro vážený SAT se zadávají do políčka „weighted SAT (parameters)“. Toto pole obsahuje skupinu celočíselných hodnot, kde první hodnota určuje velikost penalizace za každou nalezenou nesplněnou klauzuli, všechny ostatní hodnoty určují váhy jednotlivých proměnných. V případě, že tyto parametry chybí, je implicitní váha proměnné nastavena na 1, implicitní hodnota penalizace je 10.

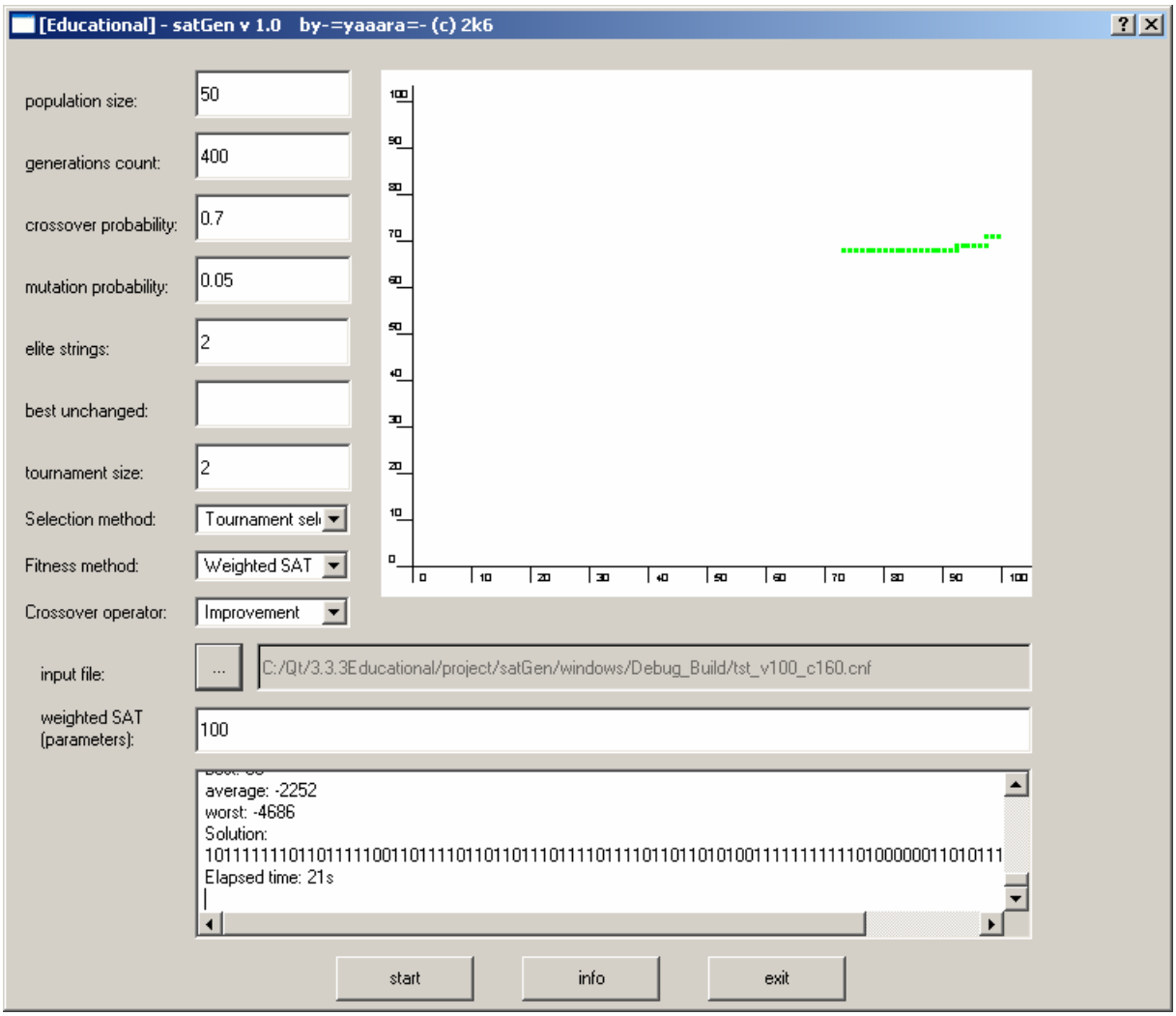
Průběh výpočtu u váženého SATu se od tří výše uvedených obrázků podstatně liší, a to v těchto konkrétních bodech:

1) ve většině případů se zobrazují pouze hodnoty nejlepšího řešení, protože průměrná a nejhorší hodnota bývá často záporné číslo a záporná čísla program nevykresluje, ale jenom vypisuje. To však závisí na hodnotě penalizace. Čím nižší hodnota penalizace, tím větší šance, že se do zobrazitelného číselného rozsahu dostane i průměrná nebo nejhorší hodnota.
2) svislá osa nereprezentuje celkový počet splněných klauzulí(v procentech) jako u standard SAT, ale určuje (opět v procentech) poměr součtu vah všech jednotkou ohodnocených proměnných jedince ku celkovému součtu vah všech proměnných(to odpovídá jedinci, který má všechny proměnné nastaveny na 1)

3) po nalezení řešení se výpočet ihned neukončí(jako u standard SAT), ale pokračuje se v hledání lépe ohodnoceného řešení, viz. příklad níže

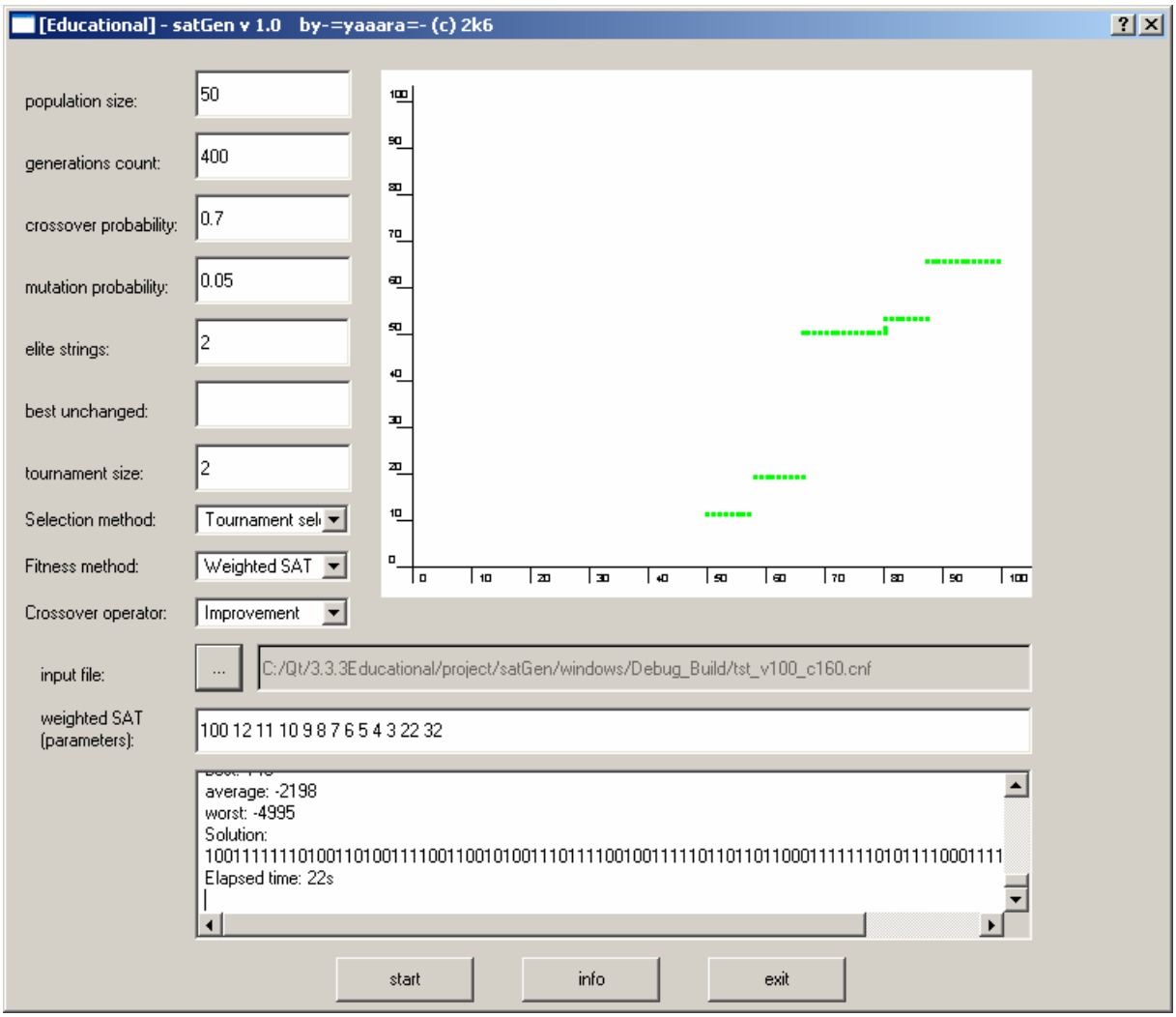
389.generation evolved
best: 67
average: -2985
worst: -4583
Solution:
11110100110010111111110001111111111111001111101110011011101101111111110111001111010010100101001011 << nalezl řešení, ale nekončí s prohledáváním

390.generation evolved
best: 69
average: -2877
worst: -4693
Solution:
10111111101101111100110111101101101110111101111011010101001111111111101000000011010111000111010111101011010111 << nalezl řešení s vahou o 2 lepší než řešení nalezené v předchozí generaci



Obr. 23 - Ukázka grafického výstupu – weighted SAT/tournament2/improvement, váhy všech proměnných nastaveny implicitně

Zkusme pozměnit několik vah proměnných z předchozího příkladu, a to následujícím způsobem - parametry váženého SATu: 100 12 11 10 9 8 7 6 5 4 3 22 32



Obr. 24 - Ukázka grafického výstupu – weighted SAT/tournament2/improvement, váhy některých proměnných nastavil uživatel

Jak je vidět, průběh výpočtu váženého SATu se poněkud změnil. V grafu vidíme „skokové změny“, které jsou pravděpodobně způsobeny změnou ohodnocení některé proměnné s vyšší vahou. Z parametrů váženého SATu vyplývá, že několik prvních proměnných má vyšší váhu než všechny ostatní. Z výpisu jednoho konkrétního řešení(viz. níže, tučně) je vidět, že na začátku je většina proměnných (s vyššími vahami) ohodnocena hodnotou 1.

351.generation evolved
best: 140
average: -2814
worst: -4783

Solution:
1001111110100110100111100110010100111011110010011111011011011000111111101011111000111100111111000110

5.2 Srovnání výkonnosti všech kombinací křížení a selekce

V této kapitole se pokusím ohodnotit výkonnost a efektivnost jednotlivých variant genetického algoritmu. Testování bude probíhat na standardních benchmarcích(staženy z Internetu, odkaz viz. níže v této kapitole) a na několika zkušebních instancích vytvořených programem randSat.

Testování algoritmů proběhne následujícím způsobem:
1) Každá implementace bude spuštěna 5 krát na každý testovací problém.
2) Doba běhu algoritmu bude ukončena v okamžiku nalezení řešení nebo po uplynutí 5000 generací nebo pokud nedojde ke změně nejlepší hodnoty po dobu 2000 generací.
3) Populace bude mít 50 jedinců, pravděpodobnost křížení je 70%, pravděpodobnost mutace je 5%, počet elitářů je stanoven na 2.

Výsledky budou zaneseny do tabulek – výsledky každého problému budou zaneseny do vlastní tabulky. Každá tabulka bude obsahovat název řešeného problému, počet proměnných, počet klauzulí, průměrné a nejlepší nalezené řešení(počet splněných a nesplněných klauzulí, celkový čas běhu algoritmu) pro danou kombinaci metod selekce a křížení.

Položka *rank* bude obsahovat pořadí metody s ohledem na úspěšnost při řešení aktuálního testu. Nakonec projdeme všechny tabulky(celkem 20 testů, tedy 20 tabulek). První místo v každé tabulce ohodnotíme třemi body, druhé místo dvěma body a třetí místo jedním bodem. Nakonec všechny body sečteme a porovnáme jednotlivé varianty. Tu variantu, která dosáhne největšího počtu bodů (největšího skóre), prohlásíme za optimální.

Pořadí důležitosti položek tabulky při určování nejlepší hodnoty:
1) average/sat
2) best/sat
3) time/avrage
4) time/best

Vybrané testy:
Testovací instance, na kterých budeme algoritmy testovat, jsou z části získány ze serveru Satlib, zbytek je vygenerován pomocí utility randSat.

Vybrané testy ze http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html:

Random-3-SAT Instances with Controlled Backbone Size (by Josh Singer)
CBS_k3_n100_m403_b10_0.cnf
CBS_k3_n100_m403_b10_999.cnf

"Flat" Graph Colouring, 3 colourable
flat30-1.cnf
flat30-99.cnf

"Morphed" Graph Colouring, 5 colourable
sw100-1.cnf
sw100-70.cnf

Planning by Henry Kautz and Bart Selman)
anomaly.cnf
bw_large.a.cnf

SAT-encoded Quasigroup (or Latin square) instances by Hantao Zhang)
qg4-08.cnf
qg7-09.cnf

SAT-encoded bounded model checking intances (by Ofer Shtrichman)
bmc-ibm-2.cnf
bmc-ibm-5.cnf

Testy vygenerované programem randSat:

tst_v10_c100.cnf
tst_v100_c160.cnf
tst_v100_c400.cnf
tst_v100_c425.cnf
tst_v25_c100.cnf
tst_v50_c500.cnf
tst_v200_c210.cnf
tst_v200_c220.cnf

Naměřené hodnoty:
viz. Příloha – část P.2

Závěrečné shrnutí výsledků:

roulette/improvement	44
tournament/improvement	36
random/improvement	35
tournament/two-point	2
tournament/uniform	2
tournament/simple	1

V levém sloupci je napsán název testované metody, v pravém sloupci je uvedeno celkové skóre, které algoritmus během provedení všech testů dosáhl.

Zjistil jsem, že nejlepších výsledků dosahuje varianta „zlepšení“ v kombinaci s výběrem jedinců pomocí rulety. Za povšimnutí stojí ještě skutečnost, že varianty umístěné na druhém(turnaj) a třetím místě(náhodný výběr) jsou téměř shodné, co se efektivity týče. Ostatní varianty jsou neefektivní a nemá smysl se jimi dále zabývat.

5.3 Vliv velikosti turnaje na efektivitu algoritmu

Zkusil jsem ještě pozorovat, jakým způsobem ovlivňuje úspěšnost řešení velikost turnaje, tj. počet jedinců, ze kterých vybíráme. Testování bylo provedeno pro skupiny o počtu 2, 4 a 8 jedinců. Pro každý test bylo spuštěno testování pro všechny metody křížení a všechny tři velikosti turnaje. Pro každý test a každou metodu byla vybrána ta velikost turnaje, která dosáhla nejlepšího řešení – toto řešení je označeno ve sloupci *rank* symbolem *x*. Nakonec jsem prošel všechny řádky ve všech tabulkách, pokud bude v řádku s tournament2 ve sloupci rank znak x, dojde ke zvýšení skóre turnaje o velikosti 2 o jedničku. Podobně pro tournament4 a tournament8.

Naměřené hodnoty:
viz. Příloha – část P.5

Výsledné skóre:

tournament2 - 6
tournament4 - 13
tournament8 - 13

Je vidět, že lepšího řešení bylo dosaženo v situaci, kdy byla použita metoda selekce turnajem z více než dvou členů. Jak metoda tournamet4, tak metoda tournament8 dosahují sice stejného, ale lepšího výsledku než v případě metody tournament2. Pokud se ale čtenář podívá na jednotlivé hodnoty ve výše uvedených tabulkách, tak si všimne, že se tyto hodnoty liší málo. Důležité však je, že tournamet4 a tournament8 překonávají tournament2.

5.3 Srovnání výkonnosti s dostupnými komerčními programy

Na závěr jsem provedl porovnání efektivity programu satGen s vybranými komerčními programy. Na internetu je možno nalézt mnoho různě efektivních programů – vybral jsme dva nejvýznamnější představitele – Satzoo 1.02 a Walksat v35. Oba programy jsem spustil s defaultními parametry, z mého programu jsem vzal všechny tři varianty s metodou křížení improvement (protože obsadily všechna tři první místa v testu měření efektivity – viz. výše).

Příklad spuštění programu Walksat:
cat ./tst/CBS_k3_n100_m403_b10_0.cnf|./walksat.exe -bad

Příklad spuštění programu Satzoo:
satzoo ./tst/CBS_k3_n100_m403_b10_0.cnf

V každém adresáři jsou připraveny testovací skripty(*test*, příp. *test.bat*), které spustí každý algoritmus několikrát na testované instance. Výstup těchto skriptů můžeme vidět v souborech *out*, příp. *out1* až *out5*

Naměřené hodnoty:
viz. Příloha – část P.3

Z naměřených výsledků je patrné, že program satGen není tak výkonný jako s ním srovnávané komerční programy. Tento rozdíl je patrný nejen v úspěšnosti v hledání řešení, ale i časových nárocích na dobu běhu algoritmu.

6 Shrnutí a závěr

Naprogramoval jsem program satGen, genetický algoritmus sloužící k řešení problému SAT. Algoritmus využívá nejen standardní operátory křížení a mutace, ale zahrnuje i další techniky urychlující výpočet. Po pečlivém prozkoumáním výsledků řešení uvedených v kapitole 5.2 je možno udělat následující závěry. Zvolil jsem si 3 metody výběru jedince a 4 metody křížení, tj. celkem 4*3 = 12 možných kombinací. Z výše uvedených grafů je zřejmé, že pro řešení problému SAT je nejvhodnější křížení metodou zlepšení (improvement). Zabývejme se dále pouze metodou zlepšení.

Co se týče metody selekce, nejlepších výsledků dosahuje metoda rulety, za ní následují metody turnaje a náhodný výběr s téměř stejným výsledkem. Jednou nevýhodou metody ruleta je skutečnost, že je nejpomalejší ze všech (všechny jedince v populaci musíme seřadit od nejkvalitnějšího k nejméně kvalitnímu). Ze zbylých dvou metod selekce vychází metoda turnaje trochu lépe než metoda náhodného výběru.

V porovnání efektivity výpočtu v závislosti na velikosti turnaje jsem zjistil (viz. kap. 5.3), že je výhodnější, když do turnaje vstupují více než dva jedinci.

V případě porovnání standardního a váženého SATu jsem si všiml rozdílného průběhu (grafu) řešení. Rozdílný výsledek byl očekáván z důvodu odlišného principu hledání řešení (viz. kapitola 4.3). Výsledky řešení lze v tomto případě ovlivňovat (narozdíl od standardního SATu) vhodnou volbou vah proměnných a hodnotou penalizace. Co se obtížnosti výpočtu týče, žádný rozdíl mezi váženým a standardním SATem jsem nepozoroval.

Velmi zajímavých výsledků jsem dosáhl v případě porovnání programu satGen s programy Walksat a Satzoo. Je vidět, že náš algoritmus není zatím schopen oběma uvedeným komerčním programům vůbec konkurovat, nejen co se týče úspěšnosti v hledání řešení, ale i dobou běhu výpočtu. Musíme si však uvědomit, že Walksat i Satzoo jsou vytvořeny na zcela odlišném principu než je princip genetických algoritmů – satGen „řeší“ najedou desítky(klidně i stovky, tisíce) instancí najednou, paralelně prohledává stavový prostor možných řešení, a proto výpočet trvá mnohem déle než u srovnávacích programů.

Co se týče efektivity, tak platí podobné závěry jako ve výše uvedeném odstavci. Výpočet pomocí genetické algoritmy(a tedy i satGen) jsou založeny víceméně na náhodě (i když metoda zlepšení se snaží tuto nevýhodu více či méně úspěšně eliminovat), naproti tomu Walksat pracuje cíleném zlepšování jedné instance řešení.

Na závěr bych rád zmínil, že pro řešení problému SAT pomocí programu satGen je nejvhodnější použít metodu výběru ruletou a křížení metodou zlepšení. Ačkoliv není současná verze programu satGen konkurenceschopná v porovnání s programy jako Walksat nebo Satzoo, speciální metody křížení a další techniky mohou v budoucnu vést k rychlejším a efektivnějším algoritmům.

7 Použité materiály a literatura

[Bla]
Complexity Theory and NP-Completeness
Alan W. Black
Coventry (Lanchester0 Polytechnic,1984

[Fuk]
Evolving Local Search Heuristics for SAT Using Genetic Programming
Alex Fukunaga
University of California, Los Angeles

[Kut]
Splnitelnost booleovských formulí
Michal Kutil
FEL ČVUT, 2005

[Dem]
Matematická logika
M. Demlová, B. Pondělíček
ČVUT Praha, 1997

[Pet]
Genetické algoritmy
Ivo Peterka
MFF UK Praha

[Jon]
Using Genetic Algorithms to Solve NP-Complete Problems
Kenneth A. De Jong , George Mason University, William M. Spears
Navy Center for Applied Research in AI

[Hao]
Evolutionary Computing for the Satisfability Problem
Jin-Kao Hao, Frédéric Lardeux, Frédéric Saubion
Université d'Angers, France

[Cmp]
SAT 2004 Competition: Solver Descriptions

[Kro]
Formal Verification #251-0247-00
Daniel Kröning
ETH Zürich, 2005/2006

[Lan]
Gary Lance
ALGORITHMS AND COMPLEXITY - Reductions

[Ssf]
Satisfability Suggested Format
revision - May 08/1993

[Wik]
http://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

[Lar]
Test Pattern Generation Using Boolean Satisfability
Tracy Larrabee

[Sta]
Kamil Staufčík
ATPG, Diplomová práce
FEL ČVUT

[Sel]
Generating Hard Satisfiability Problems
Bart Selman, David Mitchell, Hector J. Levesque

[PAA]
Materiály k předměru „Problémy a algoritmy“
Petr Fišer
http://service.felk.cvut.cz/courses/E36PAA/
FEL ČVUT

8 Příloha

P.1 Příklad vstupu a výstupu pro konkrétní data

vstup:

- soubor *tst_v10_c100.cnf* (zkráceno)

```
c
c This file was generated by -=randSat=- cnf formula generator
c
p cnf 10 100
9 1 6 0
-3 1 2 0
9 9 7 0
7 1 10 0
-5 2 9 0
-3 8 9 0
4 7 3 0
1 10 3 0
2 2 10 0
-5 7 -4 0
-9 2 -5 0
-4 5 -6 0
7 -10 9 0
4 -9 -1 0
...
```

- parametry programu satGen

Rozdíl mezi textovou a grafickou verzí programu je pouze ve formě zadání vstupních dat – vstupní data zadáme jako parametry spustitelného souboru nebo napíšeme do příslušných políček okna.

a) ukázka pro textovou variantu

zavoláme s následujícími parametry:
satGen tst_v10_c100.cnf 10 2 5000 60 1 2 3 0 2000 2
kde je 10 jedinců v generaci, 2 elitáři, vývoj trvá 5000 generací, pravděpodobnost křížení 60 %, pravděpodobnost mutace 1 %, metoda selekce = 2, metoda křížení = 3, fitness = 0, podmínka ukončení(jak dlouho se nesmí změnit nejlepší nalezená hodnota, aby došlo k předčasnému ukončení výpočtu) = 2000 generací, počet jedinců vstupujících do turnaje = 2

Níže uvedený seznam ukazuje všechny možné metody výpočtu(a jejich číselná označení):

// selection method

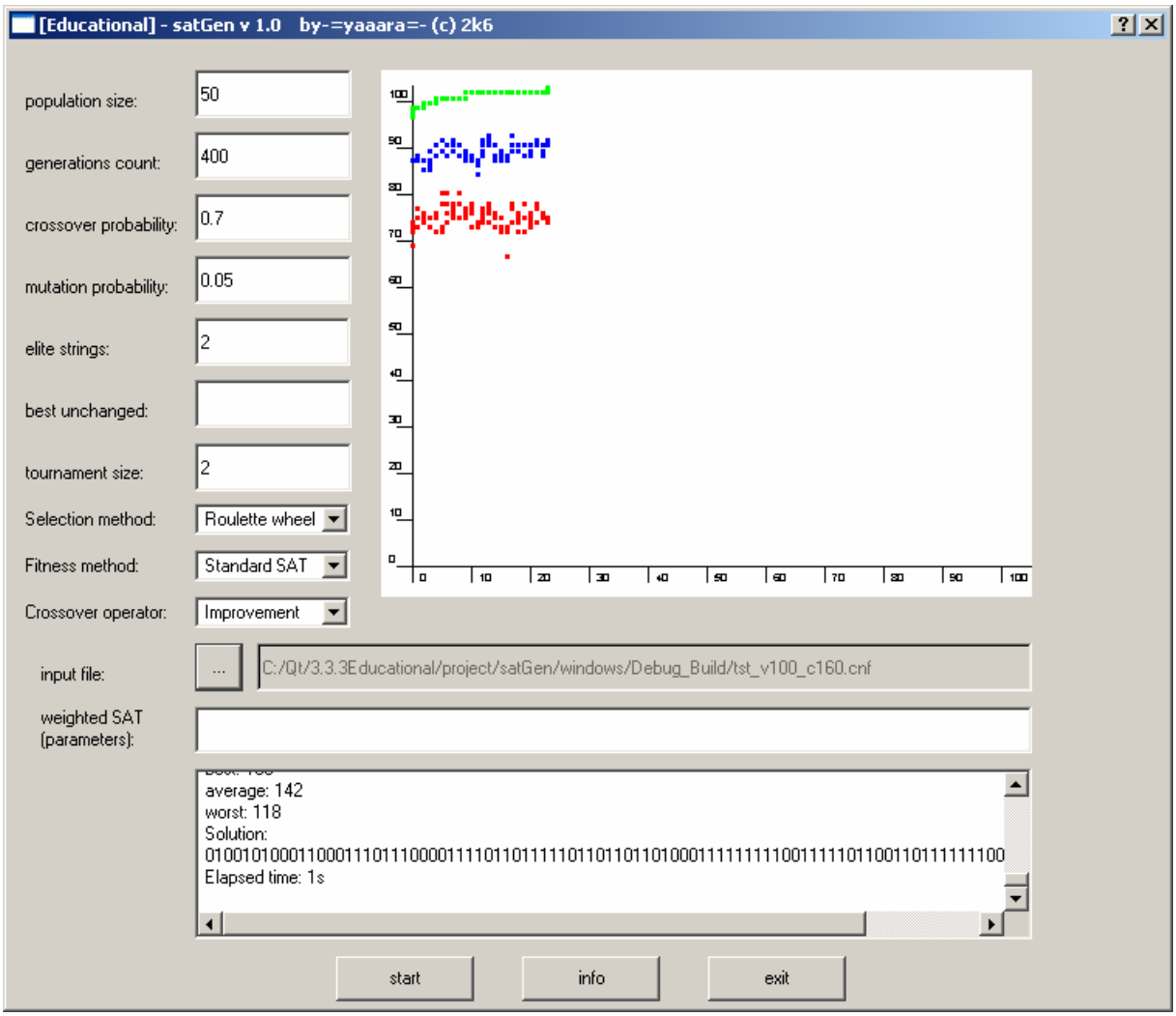
```
// 0=roulette
// 1=tournament
// 2=random

// crossover method
// 0=simple
// 1=two-point
// 2=uniform
// 3=improvement

// fitness
// 0=number of sat clauses
// 1=weighted SAT
```

výstup:

Pro každou generaci je vygenerován následující výstup:

```
1. generation evolved:
best:90                    << statistika dané populace (nejlepší,
average:86                 průměrný a nejhorší jedinec)
worst:81
```

b) ukázka pro grafickou variantu



Obr. 25 - Ukázka grafického výstupu

P.2 Výsledky testování 1

testbench name			var		cls	rank
CBS_k3_n100_m403_b10_0.cnf			100		403	
method	average			best		
	Sat	unsat	time	sat	unsat	time	
roulette/simple	366.8	36.2	25.6	365	38	26	
roulette/two-point	365.2	37.8	31.6	368	35	32	
roulette/uniform	368	35	26.2	374	29	26	
roulette/improvement	369.8	33.2	40.4	382	21	57	3
tournament/simple	363	40	31.6	367	36	32	
tournament/two-point	366.8	36.2	27.6	370	33	27	
tournament/uniform	365.8	37.2	27.6	370	33	27	
tournament/improvement	372.4	30.6	57.4	382	21	89	2
random/simple	367	36	24.4	368	35	25	
random/two-point	366.8	36.2	24.6	370	33	25	
random/uniform	365.2	37.8	24.2	367	36	24	

random/improvement	377.2	25.8	51.6	384	19	60	1

Tab. 3 – Bechmark 1

testbench name			var		cls		rank
CBS_k3_n100_m403_b10_999.cnf			100		403		
method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/simple	368.6	34.4	25.8	375	28	26	
roulette/two-point	368	35	31.6	371	32	31	
roulette/uniform	366	37	27	371	32	26	
roulette/improvement	373.4	29.6	44.8	383	20	78	2
tournament/simple	365.6	37.4	31	368	35	31	
tournament/two-point	365.2	37.8	28	367	36	28	
tournament/uniform	363.8	39.2	28	365	38	28	
tournament/improvement	374.6	28.4	48.4	380	23	59	1
random/simple	368	35	24.2	371	32	24	
random/two-point	366.2	36.8	24.4	369	34	24	
random/uniform	365.2	37.8	25.2	369	34	26	
random/improvement	372.4	30.6	38.4	383	20	53	3

Tab. 4 – Bechmark 2

testbench name			var		cls		rank
flat30-1.cnf			90		300		
method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/simple	254.8	45.2	23.2	259	41	23	
roulette/two-point	257.2	42.8	31.6	264	36	32	
roulette/uniform	258	42	23.6	261	39	24	
roulette/improvement	270	30	53.4	285	15	78	2
tournament/simple	254.4	45.6	31.8	257	43	32	
tournament/two-point	259.8	40.2	26.6	267	33	23	
tournament/uniform	259	41	23.8	262	38	24	
tournament/improvement	266.6	33.4	39.6	279	21	36	3
random/simple	256.6	43.4	24	259	41	24	
random/two-point	257.4	42.6	23.2	262	38	23	
random/uniform	258.4	41.6	23.6	262	38	23	
random/improvement	272	28	37.4	282	18	51	1

Tab. 5 – Bechmark 3

testbench name			var		cls		rank
flat30-99.cnf			90		300		
method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/simple	259.6	40.4	23.4	263	37	23	
roulette/two-point	257.4	42.6	31.4	262	38	31	
roulette/uniform	258	42	24	268	32	24	

roulette/improvement	276.4	23.6	64.8	287	13	79	1
tournament/simple	258	42	36	262	38	32	
tournament/two-point	256	44	24.6	265	35	24	
tournament/uniform	259.8	40.2	24.6	266	34	24	
tournament/improvement	273.2	26.8	35.6	281	19	60	2
random/simple	259.2	40.8	23.4	261	39	24	
random/two-point	259	41	24	266	34	24	
random/uniform	257.8	42.2	23	263	37	23	
random/improvement	270.4	29.6	49.2	281	19	59	3

Tab. 6 – Bechmark 4

testbench name			var	cls		rank
sw100-1.cnf			500	3100		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	2518.4	581.6	53.2	2561	539	52	
roulette/two-point	2494.8	605.2	55.4	2522	578	45	
roulette/uniform	2505.4	594.6	52.6	2554	546	52	
roulette/improvement	2582.4	517.6	78.2	2747	353	97	2
tournament/simple	2505.8	594.2	63.8	2516	584	64	
tournament/two-point	2495.6	604.4	62.6	2525	575	62	
tournament/uniform	2502.6	597.4	62.6	2526	574	62	
tournament/improvement	2625	475	113.8	2752	348	156	1
random/simple	2510.6	589.4	46	2563	537	42	
random/two-point	2521.4	578.6	42.4	2570	530	42	
random/uniform	2503.2	596.8	44	2528	572	44	
random/improvement	2539.6	560.4	64.4	2644	456	114	3

Tab. 7 – Bechmark 5

testbench name			var	cls		rank
sw100-70.cnf			500	3100		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	2512.2	587.8	53	2520	580	532	
roulette/two-point	2489	611	41.2	2496	604	41	
roulette/uniform	2490.2	609.8	52.2	2507	593	51	
roulette/improvement	2608.4	491.6	80	2677	423	67	1
tournament/simple	2517	583	62.8	2559	541	63	3
tournament/two-point	2517.2	582.8	63	2538	562	63	
tournament/uniform	2500.8	599.2	62	2519	581	62	
tournament/improvement	2583.6	516.4	139	2651	449	156	2
random/simple	2480	620	43.4	2505	595	43	
random/two-point	2498.8	601.2	43	2520	580	43	
random/uniform	2475.4	624.6	44.8	2493	607	47	
random/improvement	2516	584	52.6	2571	529	91	

Tab. 8 – Bechmark 6

testbench name			var		cls	rank
anomaly.cnf			48		261	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	227.8	33.2	23	232	29	23	
roulette/two-point	229.8	31.2	31.2	237	24	32	
roulette/uniform	227.8	33.2	23.6	236	25	24	
roulette/improvement	238	23	54.6	246	15	70	2
tournament/simple	230.2	30.8	31.6	232	29	32	
tournament/two-point	229	32	24	235	26	24	
tournament/uniform	228.2	32.8	24	236	25	24	
tournament/improvement	233.6	27.4	32.2	243	18	39	3
random/simple	229.4	31.6	21.8	232	29	22	
random/two-point	227	34	22	231	30	22	
random/uniform	227.6	33.4	21.8	232	29	22	
random/improvement	238.2	22.8	38	244	17	56	1

Tab. 9 – Bechmark 7

testbench name			var		cls	rank
bw_large.a.cnf			459		4675	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	3864.8	810.2	63.4	3944	731	62	
roulette/two-point	3861.4	813.6	62.6	3888	787	63	
roulette/uniform	3842.6	832.4	62.2	3874	801	62	
roulette/improvement	3906.2	768.8	101.8	3986	689	63	1
tournament/simple	3823.8	851.2	72	3849	826	71	
tournament/two-point	3844.6	830.4	87.4	3869	806	75	
tournament/uniform	3851.6	823.4	87.8	3885	790	94	
tournament/improvement	3870.6	804.4	127.8	3963	712	244	3
random/simple	3846.8	828.2	57.8	3900	775	89	
random/two-point	3859.2	815.8	50	3891	784	48	
random/uniform	3833.8	841.2	49.6	3922	753	49	
random/improvement	3894.6	780.4	100.2	3934	741	104	2

Tab. 10 – Bechmark 8

testbench name			var		cls	rank
qg4-08.cnf			512		9685	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	8093	1592	139.8	8119	1566	141	
roulette/two-point	8120	1565	123.4	8188	1497	113	
roulette/uniform	8143	1542	128.2	8193	1492	130	
roulette/improvement	8232.6	1452.4	515	8345	1340	267	2
tournament/simple	8120.8	1564.2	191.4	8168	1517	208	
tournament/two-point	8084	1601	204.8	8129	1556	197	

tournament/uniform	8115	1570	228	8181	1504	268	
tournament/improvement	8400.6	1284.4	446.4	9182	503	415	1
random/simple	8112.4	1572.6	108.4	8154	1531	127	
random/two-point	8164.6	1520.4	134.4	8242	1443	84	
random/uniform	8145.6	1539.4	117.4	8202	1483	137	
random/improvement	8179.6	1505.4	128	8275	1410	171	3

Tab. 11 – Bechmark 9

testbench name				var			cls	rank
qg7-09.cnf				729			22060	
method	average			best				
	sat	unsat	time	sat	unsat	time		
roulette/simple	18689.2	3370.8	298.2	18775	3285	329		
roulette/two-point	14926.8	7133.2	220.4	18721	3339	268		
roulette/uniform	18771	3349	280	18953	3107	291		
roulette/improvement	18804.8	3255.2	360.8	19118	2942	670	3	
tournament/simple	18776	3284	373	18851	3209	376		
tournament/two-point	18830.4	3229.6	496.4	18988	3072	525	2	
tournament/uniform	18639.2	3420.8	479	18689	3371	398		
tournament/improvement	18779.2	3280.8	524.6	19069	2991	516		
random/simple	18706.6	3353.4	217.2	18876	3184	219		
random/two-point	18723	3337	211.2	18820	3240	205		
random/uniform	18784.2	3275.8	218.2	18852	3208	196		
random/improvement	18952.6	3107.4	351.8	19167	2893	412	1	

Tab. 12 – Bechmark 10

testbench name				var			cls	rank
bmc-ibm-2.cnf				2810			11683	
method	average			best				
	sat	unsat	time	sat	unsat	time		
roulette/simple	9413	2270	176.4	9498	2185	158		
roulette/two-point	9399.4	2283.6	173.8	9440	2243	174		
roulette/uniform	9391.8	2291.2	193.6	9454	2229	184		
roulette/improvement	9502.2	2180.8	428	9542	2141	477	1	
tournament/simple	9370.4	2312.6	256	9455	2228	256		
tournament/two-point	9399	2284	256.6	9438	2245	265		
tournament/uniform	9451.2	2231.8	263.8	9542	2141	266	2	
tournament/improvement	9440.8	2242.2	420.4	9505	2178	527	3	
random/simple	9407	2276	152	9426	2257	149		
random/two-point	9396.8	2286.2	138.4	9443	2240	119		
random/uniform	9405	2278	147.4	9456	2227	157		
random/improvement	9432.6	2250.4	203.4	9507	2176	193		

Tab. 13 – Bechmark 11

testbench name	var	cls	rank
bmc-ibm-5.cnf	9396	41207	

method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/simple	32999.8	8207.2	626	33073	8134	598	
roulette/two-point	33066.4	8140.6	629.4	33165	8042	676	
roulette/uniform	33039	8168	755.8	33149	8058	703	
roulette/improvement	33370.2	7836.8	1166.2	33724	7483	2123	2
tournament/simple	32996.4	8210.6	871.4	33135	8072	860	
tournament/two-point	33196.2	8010.8	965.4	33506	7701	1218	
tournament/uniform	33099.4	8107.6	875.8	33192	8015	875	
tournament/improvement	33199.8	8007.2	1311.4	33671	7536	1431	3
random/simple	33051.6	8155.4	496.6	33208	7999	501	
random/two-point	33166.4	8040.6	498.4	33300	7907	536	
random/uniform	33065.4	8141.6	419.8	33219	7988	395	
random/improvement	33716.2	7490.8	990.2	35067	6140	916	1

Tab. 14 – Bechmark 12

testbench name		var		cls		rank
tst_v10_c100.cnf		10		100		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	93.6	6.4	31.8	95	5	33	
roulette/two-point	94.5	5.5	21.6	95	5	22	
roulette/uniform	93.4	6.4	22	96	4	22	
roulette/improvement	94.6	5.4	26.6	96	4	22	2
tournament/simple	94.2	5.6	22.2	95	5	22	
tournament/two-point	94.6	5.4	22.2	95	5	22	
tournament/uniform	93.8	6.2	22	96	4	22	
tournament/improvement	95.2	4.8	48.2	96	4	42	1
random/simple	93.8	6.2	22	95	5	22	
random/two-point	94.2	5.8	22	96	4	22	
random/uniform	94.6	5.4	31.6	96	4	31	
random/improvement	94.6	5.4	27.2	96	4	26	3

Tab. 15 – Bechmark 13

testbench name		var		cls		rank
tst_v100_c160.cnf		100		160		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	149	11	31.4	151	9	31	
roulette/two-point	150	10	24	151	9	24	
roulette/uniform	149.2	10.8	24	150	10	24	
roulette/improvement	151.8	8.2	33.2	155	5	40	2
tournament/simple	149.4	10.6	23.8	151	9	24	
tournament/two-point	149	11	28	151	9	24	
tournament/uniform	150.4	9.6	24	151	9	24	
tournament/improvement	152.2	7.8	48.2	155	5	59	1
random/simple	149.6	10.4	21.6	150	10	21	

random/two-point	149.8	10.2	22	154	6	22	
random/uniform	150.2	9.8	31.6	152	8	32	
random/improvement	150.8	9.2	26	154	6	36	3

Tab. 16 – Bechmark 14

testbench name			var		cls		rank
tst_v100_c400.cnf			100		400		
method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/simple	366.4	33.6	31.4	370	30	32	
roulette/two-point	366	34	25.4	371	29	25	
roulette/uniform	367.4	32.6	25.6	372	28	26	
roulette/improvement	377	23	49	384	16	55	1
tournament/simple	366.8	33.2	28	370	30	28	
tournament/two-point	363.2	36.8	28	365	35	28	
tournament/uniform	365	35	27.6	368	32	28	
tournament/improvement	374.4	25.6	47.6	380	20	64	3
random/simple	366	34	24.2	368	32	24	
random/two-point	366	34	24	370	30	24	
random/uniform	366.4	33.6	31.4	370	30	32	
random/improvement	376.6	23.4	38	382	18	60	2

Tab. 17 – Bechmark 15

testbench name			var		cls		rank
tst_v100_c425.cnf			100		425		
method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/simple	389.8	35.2	31.8	394	31	32	
roulette/two-point	391.2	33.8	26.2	396	29	26	
roulette/uniform	389.8	35.3	26.4	398	27	26	
roulette/improvement	399.8	25.2	58.6	411	14	65	2
tournament/simple	387.6	37.4	28.4	392	33	28	
tournament/two-point	387.6	37.4	27.8	393	32	27	
tournament/uniform	390.2	34.8	27.8	392	33	28	
tournament/improvement	400.2	24.8	67	408	17	78	1
random/simple	387.4	37.6	24.6	392	33	25	
random/two-point	389	36	24.6	393	32	25	
random/uniform	388.8	36.2	31.6	391	34	31	
random/improvement	398.4	26.6	38	405	20	52	3

Tab. 18 – Bechmark 16

testbench name			var		cls		rank
tst_v25_c100.cnf			25		100		
method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/simple	95	5	31.6	97	3	31	

roulette/two-point	95.2	4.8	22	96	4	22	
roulette/uniform	93.8	6.2	22	94	6	22	
roulette/improvement	96.2	3.8	27.2	98	2	27	1
tournament/simple	95.2	4.8	21.8	97	3	22	
tournament/two-point	95.6	4.4	21.8	97	3	22	
tournament/uniform	95.8	4.2	22	97	3	22	
tournament/improvement	96	4	38.4	98	2	48	3
random/simple	95	5	21.6	96	4	22	
random/two-point	95.6	4.4	21.2	97	3	22	
random/uniform	94.6	5.4	31	95	5	31	
random/improvement	96.2	3.8	29.4	98	2	42	2

Tab. 19 – Bechmark 17

testbench name			var	cls		rank
tst_v50_c500.cnf			50	500		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	461.8	38.2	31.6	466	34	31	
roulette/two-point	463	37	28	475	25	28	
roulette/uniform	459.6	40.4	27.8	461	39	28	
roulette/improvement	471.6	28.4	40	484	16	67	3
tournament/simple	463.2	36.8	29.4	475	25	29	
tournament/two-point	461.8	38.2	30	464	36	30	
tournament/uniform	460.8	39.2	29.6	463	37	30	
tournament/improvement	474.6	25.4	68	485	15	75	1
random/simple	456.2	43.8	25	458	42	25	
random/two-point	458.6	41.4	25.6	459	41	25	
random/uniform	463.2	36.8	31.6	472	28	32	
random/improvement	471.8	28.2	215.8	483	17	63	2

Tab. 20 – Bechmark 18

testbench name			var	cls		rank
tst_v200_c210.cnf			200	210		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/simple	194.4	15.6	31.4	196	14	31	
roulette/two-point	194	16	24	195	15	24	
roulette/uniform	195.4	14.6	23.6	198	12	23	
roulette/improvement	198.8	11.2	34.4	203	7	29	2
tournament/simple	195	15	24.2	197	13	24	
tournament/two-point	196.2	13.8	24.2	201	9	24	
tournament/uniform	195.4	14.6	23.8	197	13	24	
tournament/improvement	197.4	212.6	37.8	200	10	58	3
random/simple	154.2	55.8	23.2	195	15	23	
random/two-point	195	15	24	198	12	24	
random/uniform	194.4	15.6	31.6	196	14	31	
random/improvement	200	10	35.2	203	7	32	1

Tab. 21 – Bechmark 19

testbench name			Var	cls		Rank
tst_v200_c220.cnf			200	220		
method	average			best		
	Sat	unsat	time	sat	unsat	time	
roulette/simple	202.6	17.4	32	206	14	32	
roulette/two-point	203.4	16.6	23.2	205	15	23	
roulette/uniform	203.6	16.4	24	207	13	24	
roulette/improvement	209.4	10.6	44.2	212	8	56	1
tournament/simple	204.6	15.4	25.2	207	13	25	
tournament/two-point	202.8	17.2	25.6	205	15	25	
tournament/uniform	203.8	16.2	25.2	205	15	24	
tournament/improvement	205.8	14.2	38.8	210	10	59	3
random/simple	203.4	16.6	23	206	14	23	
random/two-point	203.6	16.4	23	207	13	23	
random/uniform	204.6	15.4	31.4	206	14	32	
random/improvement	207.6	12.4	34.6	211	9	39	2

Tab. 22 – Bechmark 20

P.3 Výsledky testování 2

testbench name			Var	cls		Rank
CBS_k3_n100_m403_b10_0.cnf			100	403		
method	average			best		
	Sat	unsat	time	sat	unsat	time	
roulette/improvement	369.8	33.2	40.4	382	21	57	
tournament/improvement	372.4	30.6	57.4	382	21	89	
random/improvement	377.2	25.8	51.6	384	19	60	3
satzoo 1.02	403	0.02	< 0.001	403	0	< 0.001	1
walksat v35	403	0.016	< 0.001	403	0	0.016	2

Tab. 23 – Srovnání 1

testbench name			var	cls		rank
CBS_k3_n100_m403_b10_999.cnf			100	403		
method	average			best		
	Sat	unsat	time	sat	unsat	time	
roulette/improvement	373.4	29.6	44.8	383	20	78	
tournament/improvement	374.6	28.4	48.4	380	23	59	
random/improvement	372.4	30.6	38.4	383	20	53	3
satzoo 1.02	403	0	< 0.001	403	0	< 0.001	1
walksat v35	403	0	< 0.001	403	0	< 0.001	1

Tab. 24 – Srovnání 2

testbench name			var		cls	rank
flat30-1.cnf			90		300	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	270	30	53.4	285	15	78	
tournament/improvement	266.6	33.4	39.6	279	21	36	
random/improvement	272	28	37.4	282	18	51	3
satzoo 1.02	300	0	< 0.001	300	0	< 0.001	1
walksat v35	300	0	< 0.001	300	0	< 0.001	1

Tab. 25 – Srovnání 3

testbench name			var		cls	rank
flat30-99.cnf			90		300	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	276.4	23.6	64.8	287	13	79	3
tournament/improvement	273.2	26.8	35.6	281	19	60	
random/improvement	270.4	29.6	49.2	281	19	59	
satzoo 1.02	300	0	< 0.001	300	0	< 0.001	1
walksat v35	300	0	< 0.001	300	0	< 0.001	1

Tab. 26 – Srovnání 4

testbench name			var		cls	rank
sw100-1.cnf			500		3100	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	2582.4	517.6	78.2	2747	353	97	
tournament/improvement	2625	475	113.8	2752	348	156	3
random/improvement	2539.6	560.4	64.4	2644	456	114	
satzoo 1.02	3100	0	< 0.001	3100	0	< 0.001	1
walksat v35	3100	0	< 0.001	3100	0	< 0.001	1

Tab. 27 – Srovnání 5

testbench name			var		cls	rank
sw100-70.cnf			500		3100	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	2608.4	491.6	80	2677	423	67	3
tournament/improvement	2583.6	516.4	139	2651	449	156	
random/improvement	2516	584	52.6	2571	529	91	
satzoo 1.02	3100	0	< 0.001	3100	0	< 0.001	1
walksat v35	3100	0	< 0.001	3100	0	< 0.001	1

Tab. 28 – Srovnání 6

testbench name	var	cls	rank

anomaly.cnf			48		261	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	238	23	54.6	246	15	70	
tournament/improvement	233.6	27.4	32.2	243	18	39	
random/improvement	238.2	22.8	38	244	17	56	3
satzoo 1.02	261	0	< 0.001	261	0	< 0.001	1
walksat v35	261	0	< 0.001	261	0	< 0.001	1

Tab. 29 – Srovnání 7

testbench name			var		cls	rank
bw_large.a.cnf			459		4675	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	3906.2	768.8	101.8	3986	689	63	3
tournament/improvement	3870.6	804.4	127.8	3963	712	244	
random/improvement	3894.6	780.4	100.2	3934	741	104	
satzoo 1.02	4657	0	0.02	4657	0	< 0.001	1
walksat v35	4675	0	0.188	4675	0	0.188	2

Tab. 30 – Srovnání 8

testbench name			var		cls	rank
qg4-08.cnf			512		9685	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	8232.6	1452.4	515	8345	1340	267	
tournament/improvement	8400.6	1284.4	446.4	9182	503	415	3
random/improvement	8179.6	1505.4	128	8275	1410	171	
satzoo 1.02			0.76			0.8	2
walksat v35	9635.7	49.3	2.031	9659	26	2.031	1

Tab. 31 – Srovnání 9

testbench name			var		cls	rank
qg7-09.cnf			729		22060	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	18804.8	3255.2	360.8	19118	2942	670	
tournament/improvement	18779.2	3280.8	524.6	19069	2991	516	
random/improvement	18952.6	3107.4	351.8	19167	2893	412	3
satzoo 1.02	22060	0	0.1	22060	0	0.1	1
walksat v35	21913.7	146.3	3.781	21924	136	3.718	2

Tab. 32 – Srovnání 10

testbench name	var	cls	rank
bmc-ibm-2.cnf	2810	11683	

method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/improvement	9502.2	2180.8	428	9542	2141	477	3
tournament/improvement	9440.8	2242.2	420.4	9505	2178	527	
random/improvement	9432.6	2250.4	203.4	9507	2176	193	
satzoo 1.02	11683	0	0.1	11683	0	0.1	1
walksat v35	11653.3	29.7	1.094	11675	8	1.094	2

Tab. 33 – Srovnání 11

testbench name		var		cls		rank
bmc-ibm-5.cnf		9396		41207		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	33370.2	7836.8	1166.2	33724	7483	2123	
tournament/improvement	33199.8	8007.2	1311.4	33671	7536	1431	
random/improvement	33716.2	7490.8	990.2	35067	6140	916	3
satzoo 1.02	41207	0	1.7	41207	0	1.7	1
walksat v35	41089.5	117.5	4.375	41112	95	4.375	2

Tab. 34 – Srovnání 12

testbench name		var		cls		rank
tst_v10_c100.cnf		10		100		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	94.6	5.4	26.6	96	4	22	
tournament/improvement	95.2	4.8	48.2	96	4	42	3
random/improvement	94.6	5.4	27.2	96	4	26	
satzoo 1.02			< 0.001			< 0.001	2
walksat v35	95.8	4.2	< 0.001	98	2	< 0.001	1

Tab. 35 – Srovnání 13

testbench name		var		cls		rank
tst_v100_c160.cnf		100		160		
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	151.8	8.2	33.2	155	5	40	
tournament/improvement	152.2	7.8	48.2	155	5	59	3
random/improvement	150.8	9.2	26	154	6	36	
satzoo 1.02	160	0	< 0.001	160	0	< 0.001	2
walksat v35	160	0	< 0.001	160	0	< 0.001	1

Tab. 36 – Srovnání 14

testbench name	var	cls	rank
tst_v100_c400.cnf	100	400	

method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/improvement	377	23	49	384	16	55	3
tournament/improvement	374.4	25.6	47.6	380	20	64	
random/improvement	376.6	23.4	38	382	18	60	
satzoo 1.02	400	0	< 0.001	400	0	< 0.001	1
walksat v35	400	0	< 0.001	400	0	< 0.001	1

Tab. 37 – Srovnání 15

testbench name	var	cls	rank
tst_v100_c425.cnf	100	425	

method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/improvement	399.8	25.2	58.6	411	14	65	
tournament/improvement	400.2	24.8	67	408	17	78	3
random/improvement	398.4	26.6	38	405	20	52	
satzoo 1.02	425	0	< 0.001	425	0	< 0.001	1
walksat v35	425	0	< 0.001	425	0	< 0.001	1

Tab. 38 – Srovnání 16

testbench name	var	cls	rank
tst_v25_c100.cnf	25	100	

method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/improvement	96.2	3.8	27.2	98	2	27	3
tournament/improvement	96	4	38.4	98	2	48	
random/improvement	96.2	3.8	29.4	98	2	42	
satzoo 1.02	100	0	< 0.001	100	0	< 0.001	1
walksat v35	100	0	< 0.001	100	0	< 0.001	1

Tab. 39 – Srovnání 17

testbench name	var	cls	rank
tst_v50_c500.cnf	50	500	

method	average			best			
	sat	unsat	time	sat	unsat	time	
roulette/improvement	471.6	28.4	40	484	16	67	
tournament/improvement	474.6	25.4	68	485	15	75	3
random/improvement	471.8	28.2	215.8	483	17	63	
satzoo 1.02			< 0.001			< 0.001	2
walksat v35	484.8	15.2	1.266	492	8	1.266	1

Tab. 40 – Srovnání 18

testbench name	var	cls	rank
tst_v200_c210.cnf	200	210	

method	average	best	

	sat	unsat	time	sat	unsat	time	
roulette/improvement	198.8	11.2	34.4	203	7	29	
tournament/improvement	197.4	212.6	37.8	200	10	58	
random/improvement	200	10	35.2	203	7	32	3
satzoo 1.02	210	0	< 0.001	210	0	< 0.001	1
walksat v35	210	0	< 0.001	210	0	< 0.001	1

Tab. 41 – Srovnání 19

testbench name			var		cls	rank
tst_v200_c220.cnf			200		220	
method	average			best		
	sat	unsat	time	sat	unsat	time	
roulette/improvement	209.4	10.6	44.2	212	8	56	3
tournament/improvement	205.8	14.2	38.8	210	10	59	
random/improvement	207.6	12.4	34.6	211	9	39	
satzoo 1.02	220	0	< 0.001	220	0	< 0.001	1
walksat v35	220	0	< 0.001	220	0	< 0.001	1

Tab. 42 – Srovnání 20

P.4 Popis jednotlivých tříd a souborů implementace

- parser.h, parser.cpp

Parser je velmi důležitá komponenta celého systému. Pomocí metody *void parseCNF(const char *inputFile, InnerForm *inner)* dojde k načtení formátu klauzule, kterou chceme vyřešit. Tato data jsou uložena do datové části objektu reprezentujícího vnitřní formu (InnerForm). V případě chybně zadaného vstupu, který není v souladu s výše popsaným formátem DIMACS CNF, je vytištěna chybová hláška a program je ukončen. Pomocí metody *void parseParameters(const char *inputData, int *weightTable, int & sumOfWeights, int numberOfVariables, int & unsatPenalty)* načítáme vstupní parametry z textového pole pro vážený SAT.

- CString.h, CString.cpp

Tyto soubory obsahují deklarace a definice dvou tříd – CString a InnerForm. Třída InnerForm reprezentuje vnitřní formu uložené klauzule. Jedná se o dvourozměné pole, kde první index značí číslo klauzule (kolikátá v pořadí daná klauzule je) a druhý index označuje číslo proměnné, která je v dané klauzuli použita.

Třída CString slouží k reprezentaci jednoho jedince v genetickém algoritmu. Jako reprezentaci každého jedince(řetězce) jsem zvolil pole hodnot typu bool s pevně danou délkou. Tato třída navíc obsahuje možné variaty hodnotící funkce (fitness) *int clausesUnsat(InnerForm *inner)*, *int clausesSat(InnerForm *inner)* a *int improvement(InnerForm *inner, int position)*, které jsou využity v genetickém algoritmu k určení kvality jedinců. Poslední významnou metodou je *int weightedSat(InnerForm *inner)*, která se stará o výpočet fitness metodou váženého SATu (s nastavitelnou penalizací, implicitně 10).

- CGeneration.h, CGeneration.cpp

Třída reprezentující vlastní genetický algoritmus. Obsahuje všechny důležité metody sloužící pro správnou funkci genetického algoritmu. Metoda *void fillRandom()* zajistí vyplnění populace náhodnými jedinci. Tato metoda je volána pouze jednou na začátku spuštění genetického algoritmu. Metoda *void selection(void)* slouží k výběru jedinců určených k selekci. K výběru je použita jedna z následujících metod, a to metoda rulety, turnaje, náhodná a zlepšení. Metoda *void crossover(double crossoverProbability, int method)* slouží k vlastnímu křížení. Ze dvou rodičů jsou výměnou jejich informací vygenerovány dva potomci, kteří jsou následně vloženy do populace. Druhým parametrem je volba metody křížení. Všechny metody křížení byly podrobně popsány v některé z výše uvedených kapitol. Metoda *void mutation(double mutationProbability)* zajišťuje náhodnou mutaci genetické informace. Metoda *void elite()* zajišťuje elitářství, kdy je skupina několika nejkvalitnějších řetězců bez nutnosti křížení přenesena do následující generace. Metoda *void printGeneration(CString** co)* vytiskne aktuální generaci na standardní výstup. Metoda *void evolution(double crossoverProbability, double mutationProbability)* zajišťuje vlastní evoluci a provádění genetického algoritmu. Jedná se o neustálé volání předem uvedených metod s danými pravděpodobnostmi úspěšnostmi provedení. Počet opakování této smyčky je explicitně zadán uživatelem. V případě že za daný počet generací není nalezeno řešení, je formule požadována za nesplnitelnou. Pro případ grafického výstupu je důležité se zmínit o dvou metodách, jejichž cílem je předat ukazatel na objekt, který zajistí zobrazení textové(případně grafické) informace v okně. Jedná se o tyto metody: *void setQMessage(QTextEdit *messageedit, QLineEdit *lineEdit9)* a *void setPaintPane(PaintPane *paintPane).*

- form1.h, form1.cpp

Třída reprezentující vlastní okno aplikace s rozmístěním jednotlivých grafických prvků.

- form1.ui.h

Zde nalezneme definici jednotlivých uživatelem vyvolaných událostí. Najdeme zde např. odpověď na otázku, co se stane, když uživatel klikne na některé tlačítko apod.

- paintpane.h

Zde nalezneme definici kreslící plochy. Tato třída obsahuje metody na vykreslení vodorovné a svislé osy, na vykreslení bodu v požadovaném místě a požadovanou barvou apod.

- interface.h

V tomto souboru definujeme způsob kompilace, tj. zda nám stačí pouze textový režim aplikace nebo vyžadujeme grafické rozhraní. Textové rozhraní lze kompilovat jak v prostředí CBuilderX, tak v prostředí QT, grafické rozhraní vyžaduje systém QT. Systém byl testován a kompilován v prostředí QT 3.3.3 Educational pod systémem Windows XP. Vzhledem k charakteru systému QT a CBuildetX je zřejmé, že aplikace může být snadno přenesena do prostředí Linuxu. Tato možnost však nebyla testována.

Poslední možností je kompilace v tzv. testovacím režimu. Testovací režim slouží k snadnějšímu testování kvality implementace genetického algoritmu. Výpočet je spuštěn

několikrát po sobě, výsledná data jsou uložena, zprůměrována a výsledek je následně vytisknut na obrazovku. Jako výsledek jsou předány hodnoty určující nejlepší nalezené řešení v sérii několika po sobě spuštěných instancí algoritmu, průměrnou hodnotu nejlepších nalezených řešení a počet uplynutých vývojových generací.

Nyní se podíváme na obsah souboru interface.h

```
//
// interface.h
//

#ifndef _interface_h
  #define _interface_h

  //#define consoleOut      << odkomentováním tohoto řádku přepneme na kompilaci do textového režimu
  //#define graphicOut      << odkomentováním tohoto řádku přepneme na kompilaci do grafického režimu
  //#define testMethod      << odkomentováním tohoto řádku přepneme na kompilaci do testovacího režimu; tento režim je totožný s režimem consoleOut až na skutečnost, že zobrazí pouze celkový nalezený výsledek a nezobrazuje průběh hledání řešení, tento režim je aktivní pouze v případě, že je odkomentován přepínač consoleOut
#endif
```

P.5 Výsledky testování 3

testbench name			var	Cls		rank
CBS_k3_n100_m403_b10_0.cnf			100	403		
Metod	average			Best		
	sat	unsat	time	sat	unsat	Time	
tournament2/simple	363	40	31.6	367	36	32	
tournament4/simple	363.8	39.2	31.8	366	37	32	
tournament8/simple	366.2	36.8	32	370	33	32	x
tournament2/two-point	366.8	36.2	27.6	370	33	27	
tournament4/two-point	367.4	35.6	31	369	34	31	x
tournament8/two-point	365.6	37.4	32	368	35	32	
tournament2/uniform	365.8	37.2	27.6	370	33	27	
tournament4/uniform	364.6	38.4	31.8	366	37	32	
tournament8/uniform	368.2	34.8	32	374	29	32	x
tournament2/improvement	372.4	30.6	57.4	382	21	89	
tournament4/improvement	373.6	29.4	59	382	21	75	x
tournament8/improvement	373.4	29.6	59.8	383	20	78	

Tab. 43 – Srovnání 21

testbench name	Var	cls	rank
CBS_k3_n100_m403_b10_999.cnf	100	403	

method	average			best			
	sat	unsat	time	sat	unsat	time	
tournament2/simple	365.6	37.4	31	368	35	31	
tournament4/simple	366.6	36.4	31.8	373	30	31	x
tournament8/simple	364	39	31	367	36	31	
tournament2/two-point	365.2	37.8	28	367	36	28	
tournament4/two-point	365.6	37.4	32	371	32	32	
tournament8/two-point	366.4	36.6	31.8	375	28	32	x
tournament2/uniform	363.8	39.2	28	365	38	28	
tournament4/uniform	366.4	36.6	31	372	31	31	x
tournament8/uniform	365	38	31.4	369	34	32	
tournament2/improvement	374.6	28.4	48.4	380	23	59	
tournament4/improvement	370.2	32.8	51.4	378	25	78	
tournament8/improvement	375.2	27.8	51.2	383	20	36	x

Tab. 44 – Srovnání 22

testbench name		Var		cls		rank
flat30-1.cnf		90		300		
method	average			best		
	sat	unsat	time	sat	unsat	time	
tournament2/simple	254.4	45.6	31.8	257	43	32	
tournament4/simple	258.2	41.8	31.4	268	32	31	x
tournament8/simple	257.4	42.6	31.4	264	36	31	
tournament2/two-point	259.8	40.2	26.6	267	33	23	
tournament4/two-point	259.4	40.6	31.2	265	35	31	
tournament8/two-point	260	40	34.4	266	34	44	x
tournament2/uniform	259	41	23.8	262	38	24	
tournament4/uniform	252.8	47.2	31.6	256	44	32	
tournament8/uniform	259.2	40.8	31.4	266	34	31	x
tournament2/improvement	266.6	33.4	39.6	279	21	36	
tournament4/improvement	259.2	40.8	42.6	263	37	60	
tournament8/improvement	273	27	58.2	284	16	77	x

Tab. 45 – Srovnání 23

testbench name		Var		cls		rank
flat30-99.cnf		90		300		
method	average			best		
	sat	unsat	time	sat	unsat	time	
tournament2/simple	258	42	36	262	38	32	
tournament4/simple	255.8	44.2	31.4	258	42	31	
tournament8/simple	259.4	40.6	31.6	262	38	31	x
tournament2/two-point	256	44	24.6	265	35	24	
tournament4/two-point	260	40	31.4	367	33	32	
tournament8/two-point	260.4	39.6	31.6	265	35	31	x
tournament2/uniform	259.8	40.2	24.6	266	34	24	x
tournament4/uniform	256	44	31.6	261	39	31	

tournament8/uniform	258.8	41.2	31.2	263	37	31	
tournament2/improvement	273.2	26.8	35.6	281	19	60	
tournament4/improvement	273.8	26.2	58	284	16	67	
tournament8/improvement	274.6	25.4	59.8	281	19	78	x

Tab. 46 – Srovnání 24

testbench name			Var	cls		rank
sw100-1.cnf			500	3100		
method	average			best		
	sat	unsat	time	sat	unsat	time	
tournament2/simple	2505.8	594.2	63.8	2516	584	64	x
tournament4/simple	2489,4	610,6	90,6	2502	598	77	
tournament8/simple	2496	604	147	2517	583	157	
tournament2/two-point	2495.6	604.4	62.6	2525	575	62	
tournament4/two-point	2509.4	590.6	97	2621	479	100	x
tournament8/two-point	2507.6	592.4	118.4	2567	533	110	
tournament2/uniform	2502.6	597.4	62.6	2526	574	62	
tournament4/uniform	2494.6	602.4	62.8	2510	590	62	
tournament8/uniform	2512.2	587.8	102.8	2570	530	94	x
tournament2/improvement	2625	475	113.8	2752	348	156	
tournament4/improvement	2569.2	530.8	158.2	2622	478	157	
tournament8/improvement	2694	406	207.2	2986	114	246	x

Tab. 47 – Srovnání 25

testbench name			Var	cls		rank
sw100-70.cnf			500	3100		
method	average			best		
	sat	unsat	time	sat	unsat	time	
tournament2/simple	2517	583	62.8	2559	541	63	
tournament4/simple	2517.4	582.6	82.2	2567	533	88	x
tournament8/simple	2506.4	593.6	170	2632	468	210	
tournament2/two-point	2517.2	582.8	63	2538	562	63	x
tournament4/two-point	2473.6	626.4	94.2	2500	600	98	
tournament8/two-point	2491.4	608.6	115.6	2545	555	100	
tournament2/uniform	2500.8	599.2	62	2519	581	62	
tournament4/uniform	2502.4	597.6	67.4	2540	560	69	x
tournament8/uniform	2487	622	114.2	2493	607	122	
tournament2/improvement	2583.6	516.4	139	2651	449	156	
tournament4/improvement	2607	493	106.2	2941	159	98	x
tournament8/improvement	2533.8	566.2	255	2586	514	362	

Tab. 48 – Srovnání 26

testbench name			Var	cls		rank
anomaly.cnf			48	261		
method	average			best		
	sat	unsat	time	sat	unsat	time	

tournament2/simple	230.2	30.8	31.6	232	29	32	x
tournament4/simple	229.6	31.4	32.6	232	29	31	
tournament8/simple	227.4	33.6	31.2	232	29	31	
tournament2/two-point	229	32	24	235	26	24	x
tournament4/two-point	228.8	32.2	31.4	235	26	31	
tournament8/two-point	228.6	32.4	31.8	231	30	31	
tournament2/uniform	228.2	32.8	24	236	25	24	
tournament4/uniform	230	31	31.4	233	28	31	x
tournament8/uniform	228.2	32.8	31.4	231	30	31	
tournament2/improvement	233.6	27.4	32.2	243	18	39	
tournament4/improvement	238.4	22.6	57.4	250	11	79	x
tournament8/improvement	235	26	50	249	12	78	

Tab. 49 – Srovnání 27

testbench name			Var		cls	rank
bw_large.a.cnf			459		4675	
method	average			best		
	sat	unsat	time	sat	unsat	time	
tournament2/simple	3823.8	851.2	72	3849	826	71	
tournament4/simple	3843.2	831.8	118.2	3863	812	162	
tournament8/simple	3872.8	802.2	198.2	3910	765	256	x
tournament2/two-point	3844.6	830.4	87.4	3869	806	75	
tournament4/two-point	3868.2	806.8	94.4	3921	754	95	x
tournament8/two-point	3834.2	840.8	152.6	3871	804	139	
tournament2/uniform	3851.6	823.4	87.8	3885	790	94	x
tournament4/uniform	3836.2	838.8	94	3874	801	94	
tournament8/uniform	3841	834	144.4	3858	817	141	
tournament2/improvement	3870.6	804.4	127.8	3963	712	244	
tournament4/improvement	4002.2	672.8	178.8	4331	344	109	x
tournament8/improvement	3954	721	305.8	4273	402	285	

Tab. 50 – Srovnání 28

P.6 Příloha na CD

/randSat	zdrojový kód programu randSat
/randSat/windows/Debug_Build/randSat.exe	zkompilovaný program randSat
/randSat/windows/Debug_Build/test.bat	ukázka použití programu randSat
/satgen_cmdline/satGen.exe	program satGen ve verzi pro příkazový řádek
/satgen_cmdline/test.bat	ukázka použití programu satGen pro příkazový řádek
/satgen_graphic/satGen.exe	program satGen v grafické verzi
/satgen_src	zdrojové kódy programu SatGen
/satgen_tester/tester.exe	program, který byl použit při testování
/satlib/benchm.html	stránka obsahující benchmarky na otestování SAT solverů, odsud byly vybrány některé testy

/satzoo/Satzoo.exe	Satzoo solver
/satzoo/test.bat	ukázka použití solveru Satzoo
/tester_src	zdrojové kódy programu tester
/text/obrazky/*.*	obrázky použité v tomto textu
/text/SAT_vX.doc	postupné vývojové verze tohoto dokumentu
/text/SAT_vX_final.doc, SAT_vX_final.pdf	finální verze tohoto dokumentu
/text/tabulkyX.doc	tabulky se zpracovanými naměřenými výsledky
/text/testy.rtf	všechny testy (vč. výsledků), které byly provedeny pomocí programu tester.exe
/text/uvod.doc, uvod.pdf	úvodní strany závěrečné zprávy (seznamy tabulek, obrázků…)
/walksat/walksat.exe	Walksat solver
/web/index.html	webová stránka projektu