Efektivní heuristika pro SAT založená na znalosti komponent souvislosti grafu problému *

Tomáš Balyo[1] a Pavel Surynek[1]

[1]Katedra teoretické informatiky a matematické logiky,
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze,
Malostranské náměstí 2/25, 118 00, Praha 1
tomas.balyo@matfyz.cz, pavel.surynek@mff.cuni.cz

Abstrakt. Článek pojednává o nové heuristice pro určování pořadí proměnných při řešení problémů booleovské splnitelnosti (SAT problémy) prohledáváním metodou DPLL. Základní ideou navržené heuristiky je použití dynamického grafu odvozeného ze stavu booleovské formule definující problém v průběhu řešení. Odvozený graf se průběhu řešení postupně mění a v každém okamžiku odráží strukturu problému tak, že části problému odpovídající jednotlivým komponentám souvislosti grafu jsou nezávislé. Heuristika pak přednostně doporučuje k ohodnocení proměnné, které patří do stejné komponenty souvislosti odvozeného grafu. Výsledkem je, že se řešící procedura omezuje při prohledávání pouze na tu část problému, která má bezprostřední vliv na splnitelnost dané komponenty, což výrazně zmenšuje prohledávaný prostor. Heuristika byla implementována v rámci experimentálního SAT řešiče, který se ukázal být konkurenceschopný vzhledem k řešičům účastnícím se soutěží SAT Competition a SAT Race.

Klíčová slova: SAT, DPLL, heuristika, souvislost, prohledávání

1 Úvod a motivace

Efektivní řešitelnost *problému booleovské splnitelnosti* (*SAT problému*) představuje jednu z hlavních otázek informatiky [7]. Problém spočívá v zodpovězení otázky, zda lze proměnné dané formule ohodnotit pravdivostními hodnotami tak, že pro toto ohodnocení je daná formule splněná. SAT problém patří mezi nejznámější zástupce třídy *NP*-úplných úloh. Za předpokladu, že $P \neq NP$, lze tento problém považovat za obecně neřešitelný pro současnou výpočetní techniku. Nicméně, v praxi se ukazuje, že jednotlivé instance problému, popřípadě celé speciální třídy SAT problémů jsou efektivně řešitelné pomocí různých pokročilých technik [19].

SAT problémy ale nepředstavují pro výzkumníky výzvu jen proto, že se jedná o teoreticky obtížné problémy. Řešení instancí SAT problémů se prakticky uplatňuje v průmyslu při verifikaci hardwaru a softwaru [21]. Další významnou praktickou aplikací je převod úloh z umělé inteligence (například problém hledání plánu) na SAT problémy [14], čímž často odpadá nutnost vyvíjet efektivní řešící software pro specifické úlohy, protože univerzální řešící systémy pro SAT problémy k dispozici již jsou.

* Podporováno z Výzkumného záměru informatické sekce MFF UK (MSM 0021620838).

V tomto článku se zabýváme návrhem nové techniky, kterou se snažíme zefektivnit proces hledání řešení SAT problémů. Konkrétně navrhujeme novou heuristiku pro řízení rozhodování řešící procedury. Heuristika je založená na znalosti komponent souvislosti jistého grafu odvozeného z problému. Jelikož klíčovou otázkou v návrhu řešících technik pro SAT problémy je vysoká efektivita případné implementace, je výpočet heuristiky prováděn s maximální úsporou času. Toho je dosaženo pomocí implicitního udržování znalosti o komponentách, což nevyžaduje další výpočty.

Navržená heuristika byla implementována v rámci experimentálního řešícího systému pro SAT problémy (SAT řešič). Ukázalo se, že dokonce bez dalších urychlujících technik, je experimentální řešič na jistých třídách problémů konkurenceschopný vůči špičkovým řešičům ze soutěží SAT Competition [15] a SAT Race [20].

V článku nejprve formálně představíme SAT problém a základní řešící metodu DPLL (oddíl 2). Poté definujeme graf odvozený z problému a na základě tohoto grafu definujeme rozhodovací heuristiku (oddíl 3). Závěrečná část je věnována experimentálnímu vyhodnocení návrhu (oddíl 4).

2 Problém booleovské splnitelnosti (SAT problém)

Booleovská formule je konečné slovo utvořené podle jistých pravidel s použitím symbolů jazyka výrokové logiky. Jazyk výrokové logiky obsahuje symboly pro výrokové proměnné (např.: x_1, x_2, x_3), logické spojky ($\neg, \wedge, \vee, \Rightarrow$) a závorky ("(", ")"). Následující definice popisuje induktivní tvorbu booleovské formule.

Definice 1 (syntaxe booleovské formule). Slovo složené z jednoho symbolu pro výrokovou proměnnou je *booleovská formule* (takovou formuli ztotožňujeme s *výrokovou proměnnou*). Jestliže jsou slova A a B booleovskými formulemi, potom slova $\neg A$, $(A \wedge B)$, $(A \vee B)$ a $(A \Rightarrow B)$ jsou rovněž booleovské formule. □

V tomto okamžiku víme, jak vypadá booleovská formule ze syntaktického hlediska, avšak otázka, která nás zajímá především, je splňování formulí. Následující definice pojem splňování formálně popisuje.

Definice 2 (sémantika booleovské formule). Nechť X je množina výrokových proměnných daného jazyka. *Pravdivostním ohodnocením výrokových proměnných* z množiny X rozumíme zobrazení $e: X \rightarrow \{True, False\}$. *Pravdivostní ohodnocení booleovských formulí* nad daným jazykem je zobrazení e^*, jež je rozšířením nějakého ohodnocení proměnných e podle následujícího induktivního předpisu: $e^*(x)=e(x)$, jestliže x je výroková proměnná; $e^*(\neg A)=True$ když $e^*(A)=False$, $e^*(\neg A)=False$ když $e^*(A)=True$; $e^*((A \wedge B))=True$, když $e^*(A)=True$ a zároveň $e^*(B)=True$, $e^*((A \wedge B))=False$ jinak; $e^*((A \vee B))=True$, když $e^*(A)=True$ nebo $e^*(B)=True$, $e^*((A \vee B))=False$ jinak; $e^*((A \Rightarrow B))=False$, když $e^*(A)=True$ a zároveň $e^*(B)=False$, $e^*((A \Rightarrow B))=True$ jinak. Booleovská formule F *má splňující ohodnocení*, jestliže existuje pravdivostní ohodnocení výrokových proměnných e takové, že $e^*(F)=True$. □

Definice 3 (problém booleovské splnitelnosti - SAT). *Problémem booleovské splnitelnosti* (*Boolean Satisfiability Problem*, SAT problém) rozumíme rozhodovací otázku, zda daná booleovská formule má nebo nemá splňující ohodnocení. □

Příklad problému booleovské splnitelnosti je ukázán na obrázku 1. Pokud je odpověď na otázku, zda daná formule má splňující ohodnocení, *ano*, zajímá nás rovněž toto splňující ohodnocení. Problém je klasickým představitelem *NP*-úplných úloh [7].

Software, který řeší problém booleovské splnitelnosti, nazýváme *úplným SAT řešičem* (dále jen *SAT řešič*; na vstupu obdrží řešič formuli, vždy skončí a na výstupu vydá odpověď a pokud je odpověď ano i splňující ohodnocení; o *neúplném* řešiči hovoříme, pokud neplatí podmínka ukončení). V současnosti existuje celá řada výkonných SAT řešičů. Mezi z vývojového hlediska nejzajímavější a výkonnostně špičkové SAT řešiče patří: *zChaff* [11, 16], *RSat* [17, 18], *MiniSat* [9, 10] a *PicoSAT* [5].

V praxi je často výhodné se při řešení problému omezit pouze na formule v *konjunktivním normálním tvaru* (*Conjunctive Normal Form*, CNF formule) [19], což je konjunkce disjunkcí literálů (obrázek 1). Tato restrikce je možná díky tomu, že pro každou booleovskou formuli existuje ekvivalentní formule v CNF tvaru [19]. Všechny zmiňované SAT řešiče očekávají na svém vstupu formuli právě ve tvaru CNF. Důvodem je jednak značná jednoduchost formulí ve tvaru CNF, což mělo za následek, že se tento tvar stal základem přenositelného formátu pro počítačový soubor obsahující booleovskou formuli (*DIMACS* formát pro booleovské formule) [8]. Dalším důvodem pro používání CNF tvaru formulí je vnitřní implementace většiny řešičů (zjednodušeně lze říci, že řešič se typicky snaží splnit co nejvíce klauzulí a proto je výhodné zadat klauzule přímo na vstupu).

SAT problém (CNF)

x_1, x_2, x_3 **... výrokové proměnné**
(nabývají hodnot *True* a *False*)

$x_1, \neg x_2, x_3$ **... literály**
(výroková proměnná nebo její negace)

$(x_1 \vee \neg x_2), (\neg x_2), (x_1 \vee \neg x_2 \vee x_3)$ **... klauzule**
(disjunkce literálů)

$(x_1 \vee \neg x_2) \wedge (\neg x_2) \wedge (x_1 \vee \neg x_2 \vee x_3)$ **... booleovská formule ve tvaru CNF**
(konjunkce klauzulí)

Řešení: formule **je** splnitelná s ohodnocením $e:\{x_1, x_2, x_3\} \rightarrow \{True, False\}$,
kde $e(x_1)=True$, $e(x_2)=False$ a $e(x_3)=True$.

Obr. 1. Problém booleovské splnitelnosti a jeho řešení s formulí ve tvaru CNF

2.1 Základní řešící metoda - DPLL

Moderní úplné algoritmy řešící problém SAT jsou téměř bez výjimky založené na algoritmu DPLL [19]. Pseudokód algoritmu DPLL je uveden jako algoritmus 1. Proces řešení problému odpovídá prohledávání prostoru všech možných ohodnocení do hloubky s několika vylepšeními.

Prvním vylepšením je včasné zastavení prohledávání. Jestliže jsou všechny klauzule splněné (řádek 7), můžeme formuli prohlásit za splněnou. Jestliže je aspoň jedna klauzule nesplněná (řádek 8), můžeme se v prohledávání vrátit o úroveň výš, protože žádné rozšíření aktuálního ohodnocení proměnných už nemůže formuli splnit.

Druhým vylepšením je tzv. heuristika čistých proměnných (*Pure Symbols Heuristic*) [19]. Proměnné, které mají pouze kladné nebo pouze záporné výskyty v dosud

nesplněných klauzulích formule, můžeme okamžitě ohodnotit tak, aby byly všechny jejich výskyty splněné. Tato heuristika je součástí algoritmu DPLL, avšak praktické experimenty ukazují, že dodatečná režie na výpočet této heuristiky je vyšší než urychlení, které přináší, proto se v dnešních SAT řešičích neimplementuje. Z tohoto důvodu heuristiku čistých proměnných v pseudokódu neuvádíme.

Posledním ale velmi důležitým vylepšením je propagace jednotkových klauzulí (*Unit Clause Propagation*) [19] (řádek 9). Jestliže se ve formuli vyskytuje klauzule, která má pouze jeden nesplněný literál (*jednotková klauzule*), je nutné tento literál (jestliže chceme, aby daná klauzule resp. celá formule byly splněné) ohodnotit pozitivně. Toto ohodnocení může vyvolat vznik dalších jednotkových klauzulí a tedy dalších takto vynucených ohodnocení.

Jestliže formule neobsahuje žádnou jednotkovou klauzuli, tj. nemůžeme hodnotu žádné proměnné odvodit, musí dojít k rozhodování. Rozhodování znamená, že si vybereme nějakou ještě neohodnocenou proměnnou a vyzkoušíme pro ni obě možnosti přiřazení *True* a *False* (řádek 4 (první rozhodnutí), řádek 10 (další rozhodnutí)).

V našem návrhu se zabýváme právě výběrem proměnné pro ohodnocení. Tento krok je v pseudokódu algoritmu zatím zapsán nedeterministicky. V článku navrhneme konkrétní způsob výběru proměnné s ohledem na co nejvyšší efektivitu.

Algoritmus 1. DPLL. Základní řešící algoritmus problémů booleovské splnitelnosti vyjádřený pomocí symbolického pseudokódu.

function DPLL-SATISFIABLE(*F*): **boolean** { *F* je booleovská formule ve tvaru CNF }
01: **let** *clauses* = množina klauzulí formule *F*
02: **let** *variables* = množina výrokových proměnných vyskytujících se v *F*
03: $e \leftarrow \varnothing$ { ohodnocení proměnných jako množina přiřazení, na začátku prázdná }
04: **let** $x \in variables$
05: **return** (DPLL(*clauses, variables,* $e \cup \{e(x)=True\}$) **or**
06: DPLL(*clauses, variables,* $e \cup \{e(x)=False\}$))

function DPLL(*clauses, variables, e*): **boolean**
07: **if** $(\forall c \in clauses)\ e^*(c)=True$ **then return** *True*
08: **if** $(\exists c \in clauses)\ e^*(c)=False$ **then return** *False*
09: $e \leftarrow e \cup$ PROPAGATE-UNIT-CLAUSES(*clauses, e*) { provedení jednotkové propagace }
10: **let** $x \in variables$ taková, že *x* nemá přiřazenu hodnotu pomocí *e*
11: **return (**DPLL(*clauses, variables,* $e \cup \{e(x)=True\}$) **or**
12: DPLL(*clauses, variables,* $e \cup \{e(x)=False\}$))

3 Heuristika založená na znalosti komponent souvislosti formule

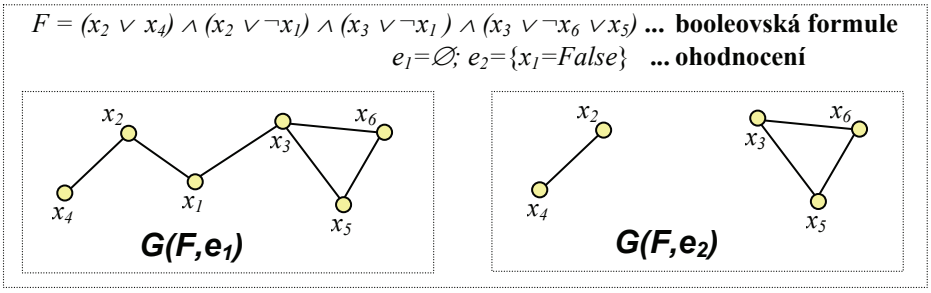
Nyní máme připravené všechny předpoklady k vyložení návrhu nové heuristiky pro výběr proměnné. Navrhovaná heuristika podstatně využívá znalosti komponent souvislosti jistého grafu odvozeného z formule.

Definice 4 (graf odvozený ze stavu ohodnocení formule - graf formule). Nechť *e* je ohodnocení některých výrokových proměnných obsažených v booleovské formuli *F*. Pro formuli *F* a ohodnocení *e* definujeme graf $G(F,e)=(V,E)$, kde $V=\{x \mid x$ proměnná formule *F* dosud neohodnocená pomocí *e*$\}$ a $E=\{\{x,y\} |$ existuje vzhledem k *e*

ještě nesplněná klauzule K v F taková, že $x \in K \wedge y \in K\}$. Graf $G(F,e)$ nazýváme *grafem formule F* při ohodnocení e. □

Právě definovaný graf má obrovský význam při efektivním řešení problému SAT (obrázek 2). Definujme *komponentu souvislosti formule* jako množinu výrokových proměnných, které odpovídají vrcholům komponenty souvislosti grafu formule (vzhledem k inkluzi maximální souvislá podmnožina vrcholů grafu). Jestliže má formule více než jednu komponentu souvislosti, můžeme ji řešit po komponentách nezávisle. Problém se tedy rozpadne na menší problémy, což má při exponenciální složitosti velký význam. Například formule, která má n proměnných se bude v nejhorším případě řešit metodou DPLL v čase $O(2^n)$. Pokud se nám ale podaří zjistit, že daná formule má například dvě komponenty souvislosti, řekněme velikosti k a l ($k+l = n$), lze podproblémy odpovídající komponentám vyřešit nezávisle v čase $O(2^k)$ resp. $O(2^l)$. Pokud navíc mají zjištěné komponenty souvislosti netriviální velikost (např.: $k,l \approx n/2$), lze dosáhnout až exponenciálního urychlení. Znalost komponent souvislosti grafu formule je tedy velmi cenná.

Všimněme si dále, že graf formule je vzhledem k průběhu řešení pomocí metody DPLL dynamický. Přesněji, pokud bychom sledovali, jak graf vypadá v každém kroku průběhu metody DPLL, pozorovali bychom, že z grafu postupně ubývají vrcholy a hrany (graf se postupně rozpadá na více komponent souvislosti). Udržování grafu formule, resp. jeho rozkladu na komponenty souvislosti v každém kroku je však výpočetně příliš náročné a SAT řešič si takovou režii na každý krok nemůže dovolit.



Obr. 2. Graf odvozený ze stavu ohodnocení formule a jeho dynamicita.

3.1 Jednotková propagace a výběr proměnné k ohodnocení

Pokusili jsme se analýzu komponent souvislosti grafu formule zabudovat do heuristiky pro výběr proměnné k ohodnocení. Místo toho abychom nákladně konstruovali graf formule v každém kroku, budeme vhodně interpretovat činnost, kterou řešící procedura DPLL provede při jednotkové propagaci. Z tohoto pak prakticky bez další režie vytěžíme potřebnou informaci o komponentách souvislosti a určíme proměnnou k ohodnocení.

Podívejme se tedy podrobněji, jak bychom mohli efektivně implementovat vyhledávání jednotkových klauzulí, tedy funkci PROPAGATE-UNIT-CLAUSES. Vždy těsně před zavoláním funkce DPLL dojde k rozhodnutí, při kterém vybereme dosud neohodnocenou proměnnou x (řádek 10) a nějak ji ohodnotíme (řádky 11 a 12). Při hledání jednotkových klauzulí stačí vyšetřovat klauzule, ve kterých má x výskyt jako

negativní literál při ohodnocení *x* hodnotou *True*, resp. jako pozitivní literál při ohodnocení *x* hodnotou *False*. V těchto klauzulích se jeden literál stane nesplněným a je možné, že tam zůstává už pouze jeden neohodnocený literál. Právě takové klauzule se stanou jednotkovými.

Během hledání jednotkových klauzulí tedy procházíme přes všechny klauzule, kde má aktuálně ohodnocená proměnná nesplněný výskyt. Při tomto procházení klauzulí potkáváme proměnné, které se v nich nacházejí. Naše heuristika bude pro ohodnocení volit vždy naposledy takto potkanou neohodnocenou proměnnou. Na základě tohoto jsme zvolili označení heuristiky *LEFV* (*Last Encountered Free Variable*). Výpočet heuristiky *LEFV* uvádíme v rámci pseudokódu funkce PROPAGATE-UNIT-CLAUSES jako algoritmus 2. Používáme globální proměnnou *lefv*, která reprezentuje naposledy potkanou neohodnocenou proměnnou. Předpokládáme, že na začátku platí *lefv*=⊥, kde ⊥ značí hodnotu nedefinováno.

Algoritmus 2. Jednotková propagace a heuristika *LEFV*. K výpočtu heuristiky *LEFV* je využit průběh jednotkové propagace.

```
function PROPAGATE-UNIT-CLAUSES(clauses, e): set
01: let l = naposledy ohodnocený literál takový, že e*(l)=True
02: Q←{l}  { Q je množina literálů, pro které je nutno provést jednotkovou propagaci }
03: while Q≠∅ do
04:    let t∈Q
05:    Q←Q-{t}
06:    for each c∈clauses taková, že c obsahuje výskyt komplementu t do
07:       if c je jednotková vzhledem k e then
08:          let u = (poslední) neohodnocený literál v c vzhledem k e  { existuje právě jeden }
09:          if u je negace výrokové proměnné x then e← e∪{x=False}
10:          else e← e∪{u=True}  { u je výroková proměnná }
11:          Q←Q∪{u}
12:       else  { c není jednotková vzhledem k e }
13:          if c obsahuje neohodnocený literál k then let lefv=proměnná příslušející k
14: return e

function SELECT-VARIABLE(variables, e): variable
15: if lefv≠⊥ and lefv není ohodnocená pomocí e then y← lefv
16: else let y∈ variables taková, že y nemá přiřazenu hodnotu pomocí e
17: lefv ←⊥
18: return y
```

Samotné zapojení navržené heuristiky do řešící procedury DPLL provedeme nahrazením řádku 10 algoritmu 1 voláním *x*←SELECT-VARIABLE(*variables, e*).

Z pseudokódu algoritmu 2 je vidět, že zaznamenávání naposledy potkané neohodnocené proměnné nepředstavuje téměř žádnou práci navíc oproti klasické jednotkové propagaci. Navíc platí, že podmínku na řádku 13 je možné implementovat tak, aby byla vyhodnocena v konstantním čase (ačkoli se zdá, že časová složitost zde bude úměrná délce klauzule). Samotné provedení rozhodnutí podle heuristiky, tedy funkce SELECT-VARIABLE má rovněž časovou složitost konstantní za předpokladu, že řádek 16 má konstantní složitost (na řádku 16 lze používat jistou sekundární heuristiku). Poznamenejme také, že co se týče paměťových nároků na heuristiku, potřebujeme pouze jednu proměnnou navíc (globální proměnnou *lefv*).

3.2 Vlastnosti heuristiky pro výběr proměnné

Na tomto místě je třeba přesně objasnit, jak souvisí popsaná heuristika *LEFV* s definovaným grafem formule a komponentami souvislosti tohoto grafu. Formálně tuto souvislost popisuje následující tvrzení.

Tvrzení 1 (vlastnosti heuristiky) Nechť K je komponenta souvislosti grafu $G(F,e)$ v okamžiku, když byla pro ohodnocení vybrána proměnná x (tj. jsme v okamžiku po vykonání řádku 10 algoritmu 1) a platí, že $x \in K$. Jestliže heuristika *LEFV* při následujícím rozhodnutí vybere neohodnocenou proměnnou y (podmínka na řádku 15 algoritmu 2 se vyhodnotí pozitivně), potom $y \in K$. ■

Ještě než tvrzení dokážeme, podejme stručný komentář. Tvrzení má formu implikace, kde na levé straně stojí podmínka, že heuristika *LEFV* vybere neohodnocenou proměnnou. Co jiného se může přihodit? Může se stát, že *LEFV* nevybere žádnu proměnnou, nebo vybere proměnnou již ohodnocenou. Ohodnocenou proměnnou podle kódu potkat nemůžeme, ale může se stát, že potkaná neohodnocená proměnná se během další propagace ohodnotí.

Důkaz: *LEFV* může vybrat jen proměnnou, kterou potká během procházení klauzulí, kde splněný literál proměnné x má komplementární výskyt. Podle definice komponenty souvislosti formule jsou potkané proměnné ve stejné komponentě jako x, protože existují klauzule, kde se x nachází společně s potkanou proměnnou - jsou to ty klauzule, kde má literál proměnné x komplementární výskyt. Jestliže se objeví jednotková klauzule a způsobí další propagaci, navštěvují se klauzule s komplementárními výskyty nějakého literálu proměnné z komponenty K, takže se opět nedostaneme mimo tuto komponentu. ■

Intuitivně tedy můžeme heuristiku *LEFV* charakterizovat tak, že se snaží řešit problém postupně po jednotlivých komponentách souvislosti grafu formule (výběr proměnných se drží v rámci komponenty, dokud není komponenta vyřešena). Tím pádem dodržujeme principy pro snížení složitosti zmíněné v úvodu tohoto oddílu.

4 Experimentální vyhodnocení

Zatím jsme teoreticky dokázali, že navržená heuristika má požadované předpoklady být efektivní heuristikou pro výběr proměnné k ohodnocení při řešení SAT problému. Nevíme však, do jaké míry bude ve skutečnosti účinná.

Aby vůbec bylo možné navrženou heuristiku otestovat, bylo třeba implementovat nový experimentální řešič SAT problémů, do nějž byla nová heuristika zabudována. Řešič byl implementován v jazyce C a byl nazván *Mei*. Zdrojový kód řešiče *Mei* a veškerá data pro reprodukování uvedených experimentů jsou dostupná na internetové stránce: http://labts.troja.mff.cuni.cz/~balyt5am/sat/mei.html. Experimentální řešič *Mei* používá jako svůj základ řešící metodu DPLL s jednotkovou propagací, přičemž vylepšením oproti DPLL je využití nové heuristiky pro výběr proměnné k ohodnocení. Implementovaný řešič tedy odpovídá pseudokódu algoritmů 1 a 2.

Cílem našich experimentů je ověřit, zda lze navrženou heuristiku považovat za úspěšnou vylepšující techniku pro SAT řešiče. Ačkoli SAT řešič *Mei* nepoužívá zda-

leka tolik vylepšujících technik jako dnešní špičkové řešiče, přesto jsme se jej rozhodli experimentálně srovnávat právě s nimi. Důvodem je, že takové srovnání je v literatuře zabývající problematikou řešení SAT problémů běžné a výsledky tak mají lepší vypovídající hodnotu.

Pro experimenty jsme zvolili existující SAT řešiče *zChaff* [11, 16], *RSat* [17, 18], *MiniSat* [9, 10] a *PicoSAT* [5]. Naše volba byla ovlivněna faktem, že tyto řešiče patří v současnosti mezi nejvýkonnější podle výsledků posledních několika ročníků soutěží SAT Competition [15] a SAT Race [20], což jsou dvě hlavní mezinárodní soutěžní fóra pro testování řešitelnosti SAT problémů. Všechny experimenty byly provedené na počítači s procesorem Intel Pentium M 1.7 GHz (2 MB L2 Cache) s operační pamětí 512 MB, pod operačním systémem GNU/Linux Debian 4.0 (jádro verze 2.6.18).

4.1 Výběr instancí SAT problémů pro experimenty

Pro experimenty jsme využili instance problémů booleovské splnitelnosti z veřejně dostupných on-line knihoven [1, 13]. Zaměřili jsme se zejména na extrémní kategorie problémů. Tím máme na mysli problémy extrémní z hlediska vnitřní struktury - zkoumali jsme problémy *vysoce strukturované* a zcela *nestrukturované* (náhodné). To jsou právě ty třídy problémů, kde očekáváme, že navržená heuristika přinese sama o sobě urychlení experimentálního řešiče *Mei* na konkurenceschopnou úroveň. Na těchto třídách problémů totiž zpravidla selhávají různé problémově závislé heuristiky a další urychlovací techniky jako jsou detekce symetrií (*Symmetry Detection*), učení se klauzulí (*Clause Learning*) a dopředný pohled (*Look Ahead*) [2], které implementují konkurenční řešiče.

Kromě zmiňovaných tříd problémů existuje také celá škála středně strukturovaných instancí problémů, které jsou typicky motivovány úlohami z průmyslové praxe (plánováním, rozvrhováním, atd.). Na této kategorii problémů však neočekáváme, že bude navržený experimentální řešič úspěšný, neboť právě zde nacházejí uplatnění zmiňované další urychlovací techniky a náš experimentální SAT řešič tak nemá šanci být konkurenceschopný, neboť žádnou z těchto technik neimplementuje.

4.2 Výsledky experimentů

Experimentální řešič *Mei* jsme testovali vzhledem k vybraným existujícím SAT řešičům na náhodných *3-SAT* problémech z oblasti fázového přechodu [6], na problémech modelujících *Dirichletův princip* (*Pigeon Holes Principle*) [13] a na problémech směrování v *FPGA* (*Field Programmable Gateway Array*) [1]. Dirichletův princip a problémy FPGA představují zástupce vysoce strukturovaných problémů.

Výsledné časy běhů testovaných SAT řešičů na náhodných instancích 3-SAT problémů z oblasti fázového přechodu jsou ukázány v tabulce 1. Problémy sestávají z náhodných klauzulí obsahující 3 literály. Přitom je poměr počtu klauzulí vůči počtu proměnných volen z *oblasti fázového přechodu* [6]. Je známo, že *těžké* náhodné SAT problémy mají tento poměr mezi hodnotami 4.0 až 5.0, což je interval označovaný jako oblast fázového přechodu. Obtížnost problémů se v této souvislosti intuitivně definuje úměrně k počtu kroků řešící metody DPLL bez vylepšení.

Tabulka 1. Časy běhů vybraných SAT řešičů na náhodných 3-SAT problémech z oblasti fázového přechodu. Byly testovány splnitelné a nesplnitelné problémy. Čas je uveden v sekundách, přičemž se jedná o celkový čas potřebný k vyřešení všech problémů z dané sady.

Náhodné 3-SAT	**Splnitelné**					**Nesplnitelné**				
Velikost prom/klauz	*75 / /325*	*100 / / 430*	*125 / / 538*	*150 / /645*	*175 / / 753*	*75 / / 325*	*100 / / 430*	*125 / / 538*	*150 / / 645*	*175 / / 753*
Počet prob.	*100*	*1000*	*100*	*100*	*100*	*100*	*1000*	*100*	*100*	*100*
MiniSat	1.17	11.59	2.05	3.32	9.18	0.95	11.85	2.45	6.07	15.80
MiniSat 2	0.70	7.26	1.47	2.79	7.19	0.45	7.78	1.91	5.10	14.04
PicoSat	1.02	10.92	1.98	3.89	10.96	0.84	11.95	2.93	7.27	22.39
zChaff	1.23	12.87	2.42	4.13	11.35	1.06	19.31	5.32	17.75	64.72
RSat 2	0.72	8.45	1.76	3.60	10.01	0.64	10.22	2.71	7.05	19.23
Mei	***1.14***	***30.12***	***11.34***	***36.15***	***165.21***	***1.32***	***66.98***	***18.4***	***52.3***	***172.70***

V rámci tohoto experimentu jsme také zjistili, že sekundární heuristika prakticky nepřijde ke slovu. Konkrétně přibližně v 99,5% rozhodnutí se použije heuristika *LEFV* (řádek 16 algoritmu 2 se vykoná jen asi v 0,5% případů).

Tabulka 2 ukazuje časy řešení získané na problémech modelující Dirichletův princip, což je otázka, zda je možné umístit $n+1$ nerozlišitelných předmětů (v originální formulaci holubů) do n přihrádek tak, že žádné dva předměty nejsou v téže přihrádce (v tabulce označeno jako hole*n*). Jedná se o nesplnitelný problém. Z teoretického hlediska je problém zajímavý tím, že pokud je k řešení použita rezoluční metoda, je pro zamítnutí existence řešení nutný důkaz alespoň exponenciální délky [12].

Tabulka 2. Časy běhů vybraných SAT řešičů na několika instancích modelujících Dirichletův princip. Všechny problémy jsou nesplnitelné. Čas je uveden v sekundách.

Dirichlet	**hole6**	**hole7**	**hole8**	**hole9**	**hole10**
Velikost prom/klauz	*42 / / 133*	*56 / / 204*	*72 / / 287*	*90 / / 415*	*110 / / 561*
MiniSat	0.04	0.21	0.67	5.44	43.18
MiniSat 2	0.03	0.19	0.64	3.39	31.24
PicoSat	0.02	0.42	1.37	15.40	313.79
zChaff	0.03	0.12	0.32	1.47	6.93
RSat 2	0.01	0.09	0.35	37.72	326.23
Mei	***0.01***	***0.10***	***0.27***	***0.55***	***1.08***

Výsledky získané na problémech směrování v FPGA jsou ukázány v tabulce 3. Problém směrování v FPGA je otázka zda existuje m disjunktních spojení skrz n propojovacích prvků (v tabulce označeno jako chnl*n*-*m*), detailnější výklad je uveden například v [1]. Opět se jedná o vysoce strukturovaný problém, přičemž obtížná varianta nastává, když $m>n$, v takovém případě problém nemá řešení. Podobně jako u Dirichletova principu potřebuje rezoluční metoda k zamítnutí existence řešení exponenciální počet kroků. Oproti Dirichletovu principu je problém FPGA považován za obtížnější.

Tabulka 3. Časy běhů vybraných SAT řešičů na několika instancích modelujících úlohu směrování v FPGA (*Field Programmable Gateway Array*). Všechny problémy jsou nesplnitelné. Čas je uveden v sekundách.

FPGA	**chnl 10-11**	**chnl 10-12**	**chnl 10-13**	**chnl 11-12**	**chnl 11-13**	**chnl 11-20**
Velikost prom/klauz	*220 / / 1122*	*240 / / 1344*	*260 / / 1586*	*264 / / 1476*	*286 / / 1472*	*440 / / 4220*
MiniSat	56.73	35.66	25.77	*>180.00*	*>180.00*	*>180.00*
MiniSat 2	25.75	75.31	161.28	*>180.00*	*>180.00*	*>180.00*
PicoSat	*>180.00*	*>180.00*	*>180.00*	*>180.00*	*>180.00*	*>180.00*
zChaff	6.88	8.27	9.73	30.58	156.80	*>180.00*
RSat 2	*>180.00*	*>180.00*	*>180.00*	*>180.00*	*>180.00*	*>180.00*
Mei	***4.24***	***4.82***	***5.62***	***8.60***	***10.11***	***23.59***

4.3 Interpretace výsledků

Získané experimentální výsledky ukazují, že navržená heuristika má vysokou účinnost. Bez dalších urychlujících technik se výkon řešiče *Mei* pohybuje mezi 10% až 100% výkonu konkurenčních řešičů na náhodných 3-SAT problémech. Přitom ztráta řešiče *Mei* narůstá směrem k větším problémům. Domníváme se, že navýšení výkonu řešiče *Mei* na plnou konkurenceschopnost vůči testovaným konkurenčním řešičům je již v možnostech kvalitnější implementace. Za povšimnutí také stojí, že se řešič *Mei* chová na náhodných problémech velmi stabilně, tj. nestává se, že by některé problémy řešil výrazně déle než jiné. Z toho vyvozujeme, že navržená heuristika neprovádí nekvalitní rozhodnutí.

Na vysoce strukturovaných problémech je ze získaných výsledků patrné, že řešič *Mei* jednoznačně vítězí. Přitom výkonnostní náskok řešiče *Mei* narůstá směrem k větším problémům. Ačkoli provedené experimenty nám nedovolují zcela obecné interpretace, lze očekávat, že účinnost navržené heuristiky se bude projevovat hlavně na problémech vyznačujících se jistou strukturou.

5 Související práce a závěr

Nejvýznamnější související prací s tímto článkem je práce [3], neboť tento článek z [3] přímo vychází. Další významnou související prací je SAT řešič *COMPSAT* [4]. Autoři tohoto SAT řešiče se rovněž zabývají otázkou, jak využít komponent souvislosti odvozených grafů, avšak tento SAT řešič zatím nedosahuje špičkových výkonů v soutěžích. Standardně se heuristiky založené na komponentách souvislosti příliš nepoužívají, neboť panuje přesvědčení, že výpočet takových heuristik je časově příliš náročný [18]. Klasickou a velmi rozšířenou heuristikou pro výběr proměnné je VSIDS (*Variable State Independent Decaying Sum*) [11, 16], tato heuristika je uplatnitelná pouze ve spojení s učením klauzulí [2] a v podstatě vybírá vždy proměnnou s největším počtem výskytů.

Navrhli jsme efektivně spočitatelnou heuristiku pro výběr proměnné k ohodnocení při řešení SAT problému metodou DPLL. Pro ověření účinnosti jsme heuristiku im-

plementovali v rámci experimentálního SAT řešiče *Mei*. Provedené experimenty ukázaly, že ačkoli řešič *Mei* nepoužívá žádné další urychlující mechanismy, je na třídě vysoce strukturovaných problémů rychlejší než špičkové SAT řešiče a na třídě náhodných problémů zůstává konkurenceschopný.

V další práci plánujeme vyvinout či převzít do experimentálního řešiče *Mei* další urychlující techniky a zlepšit implementaci, abychom řešič povýšili na konkurenceschopnou úroveň na všech třídách problémů, jak to vyžadují soutěže SAT Competition a SAT Race.

Reference

1. Aloul F. A. *Fadi Aloul's Home Page - SAT Benchmarks*. Výzkumná webová stránka, http://www.eecs.umich.edu/~faloul/benchmarks.html, University of Michigan, USA, (červen 2008).
2. Anbulagan *SAT: Algorithms and Applications*. Tutorial, the 22nd Conference on Artificial Intelligence (AAAI 2007), http://users.rsise.anu.edu.au/~anbu/slides/aaai07/SATtutoAAAI07-presented.pdf, (červen, 2008).
3. Balyo T. *Efektívny riešič problémov boolovskej splniteľnosti.* Bakalářská práce, Matematicko-fyzikální fakulta, Univerzita Karlova, Praha, 2008.
4. Biere A., Sinz C. *Decomposing SAT Problems into Connected Components*. Journal on Satisfiability, Boolean Modeling and Computation 2 (2006), 191-198, Delft University, 2006.
5. Biere A. *Picosat.* Výzkumná webová stránka, http://fmv.jku.at/picosat/, Johannes Kepler Universität Linz, Rakousko, (červen 2008).
6. Cheeseman P., Kanefsky B., Taylor W. M. *Where the Really Hard Problems Are.* Proceedings of the 12th International Joint Conference on Artificial Intelligence, (IJCAI 1991), 331-337, Morgan Kaufmann, 1991.
7. Cook S. A. *The Complexity of Theorem-Proving Procedures.* Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971), 151-158, ACM Press, 1971.
8. DIMACS. *Satisfiability Suggested Format*. Výzkumný FTP server, ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/, Rutgers University, USA, (červen 2008).
9. Eén N., Sörensson N. *An Extensible SAT-solver.* Proceedings of the Theory and Applications of Satisfiability Testing, 6th International Conference (SAT 2003), 502-518, LNCS 2919, Springer Verlag, 2003, ISBN 3-540-20851-8.
10. Eén N., Sörensson N. *The MiniSat Page*. Výzkumná webová stránka, http://minisat.se , Chalmers University, Švédsko, (červen 2008).
11. Fu Z., Marhajan Y., Malik S. *zChaff.* Výzkumná webová stránka, http://www.princeton.edu/~chaff/zchaff.html, Princeton University, USA, (červen 2008)
12. Haken A. *The Intaractability of Resolution*. Theoretical Computer Science, volume 39, 297-308, Elsevier, 1985

13. Hoos H. H., Stützle T. *SATLIB: An Online Resource for Research on SAT*. In Proceedings Theory and Applications of Satisfiability Testing, 3rd International Conference (SAT 2000), 283-292, IOS Press, 2000, http://www.satlib.org, (červen 2008).
14. Kautz H. A., Selman B. *Planning as Satisfiability*. Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92), 359-363, Austria, John Wiley and Sons, 1992.
15. Le Berre D., Roussel O., L. Simon L. *Sat Competitions*. Webová stránka soutěže, http://satcompetition.org, (červen 2008).
16. Moskewicz M. W., Madigan C. F., Zhao Y., Zhang L., Malik S. *Chaff: Engineering an Efficient SAT Solver*. Proceedings of the 38th Design Automation Conference (DAC-2001), 530-535, ACM Press, 2001, 1-58113-297-2.
17. Pipatsrisawat K., Darwiche A. *RSat*. Výzkumná webová stránka. http://reasoning.cs.ucla.edu/rsat/, University of California Los Angeles, USA, (červen 2008).
18. Pipatsrisawat K., Darwiche A. *Lightweight Component Caching Scheme for Satisfiability Solvers*. Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), 328-332, IOS Press 2004, 2004, ISBN 1-58603-452-9.
19. Russel S., Norvig P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003, USA, ISBN 0-13-790395-2.
20. Sinz C. *Sat Race 2008*. Webová stránka soutěže, http://www-sr.informatik.uni-tuebingen.de/satrace-2008/, Eberhard Karls Universität Tübingen, Německo, (červen 2008).
21. Velev M. N., Bryant R. E. *Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors*. Journal of Symbolic Computation (JSC), volume 35-2, 73-106, Elsevier, 2003.

Annotation:

An Efficient Heuristic for SAT Exploiting Connected Components of the Problem

This paper describes a new variable ordering heuristic for a Boolean satisfiability problem (SAT) solver based on search by the method DPLL. The basic idea of the heuristic is to derive a dynamic graph from the state of the problem during search. The graph reflects the structure of the problem so that parts of the problem corresponding to the individual connected components are independent. The heuristic preferably selects variables for valuation from the same connected component. The consequence is that the solving procedure restricts itself only on the part of the problem directly influencing the satisfiability of the affected component. This approach significantly reduces the space that must be searched. The heuristic was implemented within an experimental SAT solver that proved to be competitive with the solvers participating in SAT Competition and SAT Race.