Instituto de Matemática e Estatística
Universidade de São Paulo

Bacharelado em Ciência da Computação

Trabalho de Formatura Supervisionado

Resolvendo o problema PSAT com auxílio da ferramenta de software livre MiniSat

Mikail Campos Freitas

Orientador: Prof. Dr. Marcelo Finger

São Paulo

Novembro de 2012

Resumo

Os softwares livres distribuídos no projeto PSAT são usados para resolver instâncias do Problema da Satisfatibilidade Probabilística. Atualmente tais softwares são baseados no uso de softwares externos.

Buscando melhorar a performance dos softwares do projeto e também uma maior facilidade na sua distribuição e reutilização, é proposta a troca de um dos softwares externos usados assim como a inclusão do código de todos os softwares envolvidos ao código do projeto.

Sumário

I Parte Objetiva **4**

1 Introdução **5**

1.1 Contextualização 5

1.2 Objetivos 6

1.3 Fundamentos 7

1.3.1 Classes de Complexidade 7

1.3.2 Redução de Problemas e Problemas NP-completos 7

1.3.3 Problema SAT 8

2 Descrição do Problema **9**

2.1 Problema PSAT 9

3 Descrição das Soluções **11**

3.1 Redução Canônica de PSAT para SAT 11

3.2 Geração de Colunas através de Redução Turing para SAT 11

3.2.1 Base Inicial 12

3.2.2 Função Objetivo 12

3.2.3 Algoritmo 13

3.3 Geração de Colunas através de Redução Turing para Max-SAT ponderado . . 14

4 Melhorias Propostas **15**

5 Trabalho Realizado **17**

5.1 Implementação original 17

5.1.1 Chamadas ao zChaff 17

5.1.2 Chamadas ao LA 17

5.1.3 Chamadas ao wmaxsatz 18
5.2 Implementação modificada 18

6 Testes **20**
6.1 Instâncias Satisfazíveis 20
6.2 Instâncias Insatisfazíveis 24
6.3 Instâncias Aleatórias 28

7 Conclusão **30**

II Parte Subjetiva **31**

1 Agradecimentos **32**

2 Desafios e frustrações **32**

3 Relação com matérias do curso **33**

Referências e Bibliografia **34**

I

Parte Objetiva

1 Introdução

1.1 Contextualização

O problema da Satisfatibilidade Probabilística (abreviado PSAT) é uma extensão do problema da Satisfatibilidade Booleana (SAT), onde é combinada dedução lógica e probabilística. Nele temos um conjunto de fórmulas proposicionais e uma probabilidade associada a cada fórmula. A solução de uma instância do PSAT é decidir se esse conjunto de restrições de probabilidades, aplicado ao conjunto de fórmulas associadas, é consistente ou não.

Sua primeira formulação é creditada a George Boole, em 1854[1], e desde então foi algumas vezes redescoberto separadamente[7, 8]. Em 1986 Nilsson reintroduziu o problema para a comunidade de Inteligência Artificial[13] e em 1988 foi demonstrado ser NP-completo[6].

> **Exemplo 1** (instância PSAT): Três amigos costumam ir a um bar, de modo que em todas as noites pelo menos dois deles estão em sua mesa de costume. Entretanto, cada um afirma ir ao bar "apenas" 60% das noites. Eles estão falando a verdade?

Devido a sua característica não-determinística o PSAT requer métodos mais complexos para ser resolvido, como métodos de álgebra linear e técnicas de programação linear aplicadas ao algoritmo simplex, em comparação ao SAT, para o qual a maior parte das soluções é baseadas no algoritmo DPLL[3, 12] (que emprega backtracking no espaço de possíveis valorações para as variáveis).

O PSAT é de crucial importância para modelagem lógico-probabilística de problemas e também tem aplicação em aprendizado automático, linguística computacional, verificação de software e hardware, entre outras.

Nesse trabalho serão discutidos métodos que procuram abordagens baseadas em lógica e fazem uso de outras soluções existentes (resolvedores SAT) para obter soluções eficientes para o PSAT.

1.2 Objetivos

Há três softwares livres que resolvem o PSAT (resolvedores PSAT) disponibilizados no projeto PSAT[1], originalmente desenvolvidos por Marcelo Finger e Glauber de Bona. Os objetivos desse trabalhos são:

- melhorar a performance dos resolvedores do projeto PSAT
- criar métodos para comparação da eficiência desses diferentes resolvedores
- facilitar a distribuição e reutilização dos softwares do projeto

[1] `http://sourceforge.net/projects/psat/files/`

1.3 Fundamentos

1.3.1 Classes de Complexidade

A Teoria da Complexidade Computacional destina-se a classificar problemas computacionais de acordo com a sua dificuldade (i.e. a complexidade da sua solução). O tipo dos problemas classificados são Problemas de Decisão, nos quais para cada instância do problema, a respectiva resposta é *sim* ou *não* (também representada como *1* ou *0*).

Algumas classes de complexidade fundamentais são:

- *classe P*: problemas solúveis em tempo polinomial em relação ao tamanho da entrada.
- *classe NP*: problemas para os quais uma resposta *sim* pode ser certificada em tempo polinomial.
- *classe co-NP*: problemas para os quais uma resposta *não* pode ser certificada em tempo polinomial.

1.3.2 Redução de Problemas e Problemas NP-completos

Dizemos que um problema Q pode ser *reduzido polinomialmente* a um problema Q' (notação: $Q \leq_p Q'$) quando conseguimos uma transformação T tal que

(1) se X é uma instância de Q então $T(X)$ é uma instância de Q',

(2) X e $T(X)$ têm a mesma resposta e

(3) T consome tempo polinomial no tamanho de X.

Desse modo, se conseguimos resolver Q' em tempo polinomial, ao aplicarmos a transformação, que também é polinomial, conseguimos resolver Q em tempo polinomial. Assim, $Q' \in P \Rightarrow Q \in P$.

Uma *redução de Turing* de Q' para Q é um algoritmo que resolve o problema Q' aplicando uma redução de Q' para Q e utilizando um oráculo para resolver o problema Q.

Além disso, um problema Q é dito *NP-completo* quando

(1) $Q \in NP$ e

(2) $Q' \leq_p Q$, para todo Q' em *NP*.

1.3.3 Problema SAT

Definições:

- *átomo* é uma variável pertencente a $\{0, 1\}$ (e.g. p, q, r).
- *literal* é um átomo, ou a negação de um átomo (e.g. p, $\neg q$).
- *cláusula* é uma disjunção de literais (e.g. $\neg p$, $\neg q \vee r$).
- *fórmula* é uma conjunção de cláusulas (e.g. $p \wedge (q \vee \neg r)$, $a \wedge (\neg b \vee \neg c) \wedge d$).

Um dos problemas de decisão mais importantes para a área de Complexidade Computacional é o Problema da Satisfatibilidade Booleana (abreviado SAT). Ele constitui-se em decidir se uma dada fórmula booleana pode ser satisfeita ou não (i.e. se é possível encontrar uma valoração para o conjunto de variáveis da fórmula de forma que a mesma seja verdadeira).

> **Exemplo 2:** Dada $F(p, q, r) = (p \vee \neg q) \wedge (q \vee \neg r) \wedge (\neg p \vee \neg q \vee r) \wedge r$.
> É possível satisfazer a fórmula F?

Sua importância na área Complexidade Computacional é ter sido o primeiro problema demonstrado ser NP-completo, por Cook[2]. Dessa forma sabemos que qualquer problema pertencente à classe NP pode ser reduzido em tempo polinomial para o SAT. Não só isso mas outras aplicações importantes do SAT são: planejamento em inteligência artificial, verificação de equivalência combinacional, softwares de projeto de circuitos integrados, verificação de software, depuração, entre outras.

Devido ao grande número de aplicações do SAT, diretas e indiretas (já que sabemos que diversos problemas podem ser, e de fato são, reduzidos ao SAT), softwares para resolvê-lo têm evoluído muito nos últimos anos, de modo que, por exemplo, implementações atuais são mais de 30 vezes mais rápidas do que no ano de 2001[10]. Algumas competições bianuais foram estabelecidas para comparar a implementação de soluções: SAT Competition[1], SAT-Race[2] e SAT Challenge[2].

Um software resolvedor SAT que merece destaque é o MiniSat[3][3]. Foi premiado em duas de quatro edições em que participou do SAT Competition, inclusive incorporando-se em 2009 à competição (nova categoria: *Best Minisat Hack*), e foi premiado em todas as edições do SAT-Race.

[1] `http://www.satcompetition.org/`
[2] `http://baldur.iti.uka.de/`
[3] `http://minisat.se/`

2 Descrição do Problema

2.1 Problema PSAT

É apresentada aqui a definição do problema, segundo Finger e de Bona[4]:

Consideremos o conjunto de variáveis booleanas $X = \{x_1, ..., x_n\}$ e seja $\mathcal{P}$ o conjunto de todas as fórmulas proposicionais sobre essas variáveis. Inicialmente definimos uma valoração v como uma atribuição de *verdadeiro* ou $falso$ a uma dessas variáveis, i.e. $v : X \to \{0, 1\}$. Em seguida estendemos a definição para fórmulas inteiras de forma que $v : \mathcal{P} \to \{0, 1\}$ (e.g. sejam α e β fórmulas proposicionais, $v(\alpha \vee \beta) = 1$ sse $v(\alpha) = 1$ ou $v(\beta) = 1$).

Seja $V = \{v_1, ..., v_{2^n}\}$ o conjunto de todas as possíveis valorações sobre X. Definimos a distribuição de probabilidade sobre valorações proposicionais como a função $\pi : V \to [0, 1]$ de modo que $\sum_{i=1}^{2^n} \pi(v_i) = 1$. Assim, a probabilidade de uma fórmula proposicional $\alpha \in \mathcal{P}$ segundo a distribuição π é $P_\pi(\alpha) = \sum\{\pi(v) \mid v(\alpha) = 1\}$.

Uma instância do problema PSAT é um conjunto $\Sigma = \{P(\alpha_i) \bowtie_i p_i \mid 1 \leq i \leq k\}$, onde $\alpha_1, ..., \alpha_k \in \mathcal{P}$ são fórmulas proposicionais restritas às $p_1, ..., p_k$ probabilidades associadas segundo as relações $\bowtie_1, ..., \bowtie_k \in \{=, \leq, \geq\}$.

O problema PSAT consiste em *decidir* se existe uma distribuição π que satisfaça todas as restrições para uma instância Σ.

Exemplo 3 (formulação): Considerando que os amigos do Exemplo 1 são representados pelas variáveis booleanas x_1, x_2 e x_3, uma formulação para essa instância é $\alpha = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$, $P(\alpha) = 1, P(x_1) = P(x_2) = P(x_3) = 0.6$.

Dizemos que uma instância PSAT Σ está na *forma normal atômica* quando ela pode ser particionada em dois conjuntos (Γ, Ψ) de forma que $\Gamma = \{P(\alpha_i) = 1 \mid 1 \leq i \leq m\}$ e $\Psi = \{P(y_i) = p_i \mid y_i \text{ é um átomo e } 1 \leq i \leq k\}$. Isso é, Γ é a instância SAT da forma normal e Ψ é o conjunto de probabilidades atribuídas.

Exemplo 4 (forma normal atômica): Na formulação do Exemplo 3 temos a fórmula $\alpha = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$ que é equivalente à fórmula $(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3)$, onde temos a conjunção de três cláusulas que precisam ser satisfeitas para que α seja. Como as probabilidades associadas já são atribuídas diretamente a átomos, uma forma normal atômica para essa instância é $\Gamma = \{(x_1 \vee x_2), (x_1 \vee x_3), (x_2 \vee x_3)\}, \Psi = \{P(x_1) = P(x_2) = P(x_3) = 0.6\}$.

Segundo Finger e de Bona[4] para qualquer instância Σ do PSAT há uma instância associada $\langle \Gamma, \Psi \rangle$ na forma normal atômica, de forma que Σ é satisfazível se e somente se $\langle \Gamma, \Psi \rangle$ é satisfazível. Além disso, tal instância pode ser construída em tempo polinomial.

Dizemos que uma valoração v sobre $y_1, ..., y_k$ é Γ-*consistente* quando podemos estender v para $y_1, ..., y_k, x_1, ..., x_n$ de modo que $\Gamma(v) = 1$. Desse modo, também dizemos que o vetor $[1 \ v(y_1) \ ... \ v(y_k)]^t$ é Γ-*consistente*.

Lema 1:[4] *Uma instância* $\langle \Gamma, \Psi \rangle$ *na forma normal atômica é satisfazível sse existe uma matriz* A_Ψ *de dimensão* $k+1 \times k+1$ *onde todas as suas colunas são* Γ-*consistentes e* $A_\Psi \pi = p, \sum \pi = 1$ *tem solução* $\pi \geq 0$.

$$A\pi = p \ \Rightarrow \ \begin{bmatrix} 1 & \cdots & 1 \\ a_{1,1} & \cdots & a_{1,k+1} \\ \vdots & \ddots & \vdots \\ a_{k,1} & \cdots & a_{k,k+1} \end{bmatrix} \cdot \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_{k+1} \end{bmatrix} = \begin{bmatrix} 1 \\ p_1 \\ \vdots \\ p_k \end{bmatrix} \tag{1}$$

$$a_{i,j} \in \{0, 1\}, \qquad A \text{ é não-singular}, \qquad \pi_j \geq 0$$

Uma matriz A que satisfaz as condições (1) acima é dita *solução viável* para $\langle \Gamma, \Psi \rangle$.

Sem perda de generalidade a partir de agora assumiremos que as probabilidades $p_1, ..., p_k$ em (1) estão em ordem descrescente.

3 Descrição das Soluções

A maior parte das soluções apresentadas para o PSAT antes de [4] são baseadas apenas em métodos algébricos. Procurando utilizar o fato de que soluções atuais para o SAT são já muito eficientes, Finger e de Bona propuseram e implementaram[4, 5] três soluções que utilizam lógica, baseadas em um oráculo SAT. Tais soluções serão descritas brevemente nessa seção.

3.1 Redução Canônica de PSAT para SAT

A primeira solução proposta é uma tradução direta de uma instância PSAT para uma instância SAT. Essa tradução é feita através da representação binária das variáveis e operações envolvidas em (1).

Considerando que a precisão em questão é de b bits, podemos reescrever os vetores π e p como um conjunto de variáveis que representam os seus bits:

$$\begin{bmatrix} 1 & \cdots & 1 \\ a_{1,1} & \cdots & a_{1,k+1} \\ \vdots & \ddots & \vdots \\ a_{k,1} & \cdots & a_{k,k+1} \end{bmatrix} \cdot \begin{bmatrix} \pi_1^b ... \pi_1^1 \\ \pi_2^b ... \pi_2^1 \\ \vdots \\ \pi_{k+1}^b ... \pi_{k+1}^1 \end{bmatrix} = \begin{bmatrix} 1 \\ p_1^b ... p_1^1 \\ \vdots \\ p_k^b ... p_k^1 \end{bmatrix}$$

Onde a codificação é feita da seguinte forma:

- operações de multiplicação de elementos $a_{i,j}$ por π_j são codificadas como conjunção dos bits de π_j com $a_{i,j}$
- operações de soma são codificadas como soma binária
- operações de igualdade são codificadas como equivalência binária

É construída uma instância SAT que corresponde a esse conjunto de codificações juntamente com a restrição de que as colunas de A_Ψ devem ser Γ-consistentes e depois é usado o resolvedor SAT para essa instância construída. Com as variáveis $\pi_j^b, ..., \pi_j^1$, $1 \leq j \leq k+1$, reconstruímos o vetor π, que é a solução para a instância PSAT.

3.2 Geração de Colunas através de Redução Turing para SAT

O intuito desta e da próxima solução (subseção 3.3) é utilizar o algoritmo Simplex da programação linear em conjunto com o método de geração de colunas baseada no uso de um oráculo. Será descrita agora a solução que usa um oráculo SAT para a geração de colunas.

3.2.1 Base Inicial

Começamos a solução construindo uma *instância relaxada* para a instância PSAT $\langle \Gamma, \Psi \rangle$ na forma normal atômica em questão, onde é ignorado o conjunto Γ de restrições. Isso é, basta encontrarmos uma matriz que satisfaça (1) para termos uma solução para a instância relaxada.

Considere a matriz triangular superior unitária $A_{(0)}$ de dimensão $k+1 \times k+1$ e a expressão abaixo:

$$A_{(0)}\pi_{(0)} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} \cdot \pi_{(0)} = \begin{bmatrix} 1 \\ p_1 \\ \vdots \\ p_k \end{bmatrix} \tag{2}$$

Dizemos que $A_{(0)}$ é a base inicial para o nosso problema.

É fácil ver que sempre existe uma solução $\pi_{(0)}$ para (2) e que para obtermos tal solução basta aplicarmos uma vez o método de substituição para trás no sistema linear equivalente. Como sabemos que $p_i - p_{i+1} \geq 0$, então também temos que $\pi_j \geq 0$.

Apesar de $A_{(0)}$ e $\pi_{(0)}$ serem base e solução para (2), as colunas de $A_{(0)}$ podem não ser Γ-consistentes. Desse modo, precisamos de um algoritmo que itere sobre essa base, de modo que no fim de sua execução tal matriz tenha apenas colunas Γ-consistentes, resolvendo assim a instância PSAT.

3.2.2 Função Objetivo

Consideremos o conjunto $J = \{j \mid A^j$ é a j-ésima coluna de A, A^j é Γ-inconsistente$\}$ e a soma f das probabilidades associadas aos índices de J, $f = \sum_{j \in J} \pi_j$.

Definimos nossa função objetivo como $\langle |J|, f \rangle$, combinando o número de colunas Γ-inconsistentes em cada passo com a soma das probabilidades associadas a essas colunas. É fácil ver que as duas componentes dessa função se tornam zero ao mesmo tempo.

Também definimos que um valor $\langle |J'|, f' \rangle$ é *reduzido* em relação a outro valor $\langle |J|, f \rangle$ quando

(1) $|J'| < |J|$ ou

(2) $|J'| = |J|$ e $f' < f$.

Assim temos, semelhante a um problema linear, o problema:

$$\begin{aligned} \min \quad & \langle |J|, f \rangle \\ \text{sujeito a} \quad & A\pi = p \\ & \pi \geq 0 \\ & J = \{j \mid A^j \text{ é } \Gamma\text{-inconsistente}\} \\ & f = \sum_{j \in J} \pi_j \end{aligned} \tag{3}$$

3.2.3 Algoritmo

Precisamo de um algoritmo que em cada passo gere uma coluna que reduza a função objetivo. Tal coluna é gerada com o procedimento *geraColuna* que, usando um resolvedor SAT, gera uma coluna Γ-consistente a qual tentamos fazer substituir uma coluna Γ-inconsistente na base corrente, reduzindo $|J|$ e assim a função objetivo.

Caso não seja possível substituir uma coluna Γ-inconsistente na base corrente a coluna gerada garante que f será reduzida ao substituirmos uma coluna Γ-consistente por ela.

Caso não exista uma coluna que reduza a função objetivo, o procedimento *geraColuna* devolve o valor *INSAT*.

Entrada: Instância PSAT na forma normal atômica $\langle \Gamma, \Psi \rangle$
Saída: Não, caso $\langle \Gamma, \Psi \rangle$ seja insatisfazível; caso contrário, matriz A que satisfaz (2).

$p \leftarrow ordenaDecrescendo(\{1\} \cup \{p_i \mid P(y_i) = p_i \in \Psi, 1 \leq i \leq k\})$;
$A_{(0)} \leftarrow$ base inicial de dimensão $k+1$; `// instância relaxada`
$c \leftarrow 0$, calcula $\langle |J|_{(c)}, f_{(c)} \rangle$;
enquanto $\langle |J|_{(c)}, f_{(c)} \rangle \neq \langle 0, 0 \rangle$ **faça**
    $b_{(c)} \leftarrow geraColuna(A_{(c)}, p, \Gamma)$;
    se $b_{(c)} = INSAT$ **então**
        retorna *Não*; `// instância insatisfazível`
    senão
        $A_{(c+1)} \leftarrow substitui(A_{(c)}, b_{(c)})$;
        $c \leftarrow c + 1$, recalcula $\langle |J|_{(c)}, f_{(c)} \rangle$;
    fim
fim
retorna $A_{(c)}$; `// instância satisfazível`

Algoritmo 3.2.1: PSAT através de Geração de Colunas

A instância construída e submetida ao resolvedor SAT reflete a restrição de Γ-consistência para a coluna gerada assim como também as restrições algébricas para que essa coluna possa

substituir alguma coluna de A. Uma descrição completa sobre como essa instância é gerada e sobre a implementação do procedimento *geraColuna* pode ser encontrada em [4].

O Algoritmo 3.2.1 ilustra a solução desejada. Nele são feitos múltiplos usos de um resolvedor SAT: chamado $k+1$ vezes para calcular o número inicial de colunas Γ-inconsistentes na linha 3; e chamado no máximo $k+1$ vezes em cada execução do procedimento *geraColuna* na linha 5.

Como podemos ver há dois casos de parada para algoritmo: caso em algum passo não seja possível gerar uma coluna que reduza a função objetivo e ainda há alguma coluna Γ-inconsistente em A, então a instância PSAT é insatisfazível; caso a função objetivo se torne $\langle 0, 0\rangle$, i.e. foi encontrada uma base A com todas as colunas Γ-consistentes, então A é solução para (2) e, assim, para a instância em questão.

3.3 Geração de Colunas através de Redução Turing para Max-SAT ponderado

Na solução anterior o procedimento *geraColuna* foi usado para gerar uma coluna qualquer Γ-consistente com custo associado negativo, i.e. de modo a reduzir o valor da função objetivo.

Alternativamente, ao invés de gerarmos uma coluna com custo associado negativo, podemos querer gerar a coluna com o *menor* custo associado. Desse modo garantimos que a coluna gerada em cada passo é a que mais reduz a função objetivo, atingindo o valor $\langle 0, 0\rangle$ o mais rápido possível.

Podemos gerar tal coluna com o auxílio de um resolvedor Max-SAT Ponderado[9], problema no qual temos um conjunto de cláusulas com pesos associados e queremos a valoração que tenha o maior peso associado (de cláusulas satisfeitas). A diferença entre essa solução e a solução anterior é de que em cada passo do algoritmo, para o procedimento *geraColuna*, é construída uma instância Max-SAT Ponderado que corresponda à restrição de Γ-consistência da coluna a ser gerada, assim como à restrição de menor custo associado a essa coluna. A descrição exata de como tal instância é construída é apresentada em [5].

Desse modo, o que difere nessa solução da solução anterior é que no Algoritmo 3.2.1 o procedimento *geraColuna* na linha 5 faz uma chamada ao resolvedor Max-SAT Ponderado, ao invés de diversas chamadas ao resolvedor SAT.

4 Melhorias Propostas

Como visto, a atual implementação dos resolvedores PSAT é baseada no uso de um oráculo para o problema SAT. Para isso é usado o software resolvedor SAT zChaff[1][12], que possui licença restrita para uso em pesquisa e não possibilita redistribuição sem consenso da Universidade de Princeton[2]. Com isso há alguns problemas para o projeto PSAT: usuários interessados no projeto devem obter o zChaff separadamente, que tem disponibilidade garantida apenas pela Universidade de Princeton; devido a dificuldade de distribuição do zChaff, o projeto é distribuído sem ele e assim o software é utilizado externamente; usuários interessados têm a mesma restrição de uso do zChaff, já que o projeto depende dele.

Além do zChaff, também são usados outros dois softwares no projeto: wmaxsatz[3] (resolvedor Max-SAT Ponderado) e LA (inversor de matrizes, atualmente só distribuído no projeto PSAT). Ambos os softwares possuem licença GPLv3[4], que não impõe restrição sobre o seu uso nem sobre a modificação e distribuição do seu código fonte.

Com o uso desses três softwares adicionais são introduzidas algumas ineficiências como: a interface e troca de dados de cada um desses softwares com os softwares do projeto é feita através de arquivos, o que resulta em diversas operações de leitura e escrita em disco rígido, que consomem tempo muito elevado em relação à memória RAM; e gasto excessivo de tempo na execução de cada um desses softwares no momento em que são requisitados no projeto, onde são envolvidas diversas trocas de contexto por parte do sistema operacional devido à criação de novos processos.

O resolvedor SAT MiniSat além de, como já mencionado, ter demonstrado desempenho excepcional, foi desenvolvido com o intuito de facilitar sua integração em outros projetos[5] e possui licença MIT[6], que não impõe restrição sobre o seu uso, modificação nem distribuição e é compatível com a licença GPLv3.

[1] `http://www.princeton.edu/~chaff/zchaff.html`
[2] `http://www.princeton.edu/~chaff/zchaff/COPYRIGHT`
[3] `http://home.mis.u-picardie.fr/~cli/EnglishPage.html`
[4] `http://www.gnu.org/licenses/gpl-3.0.html`
[5] `http://minisat.se/Main.html`
[6] `http://opensource.org/licenses/MIT`

Desse modo são propostas as seguintes modificações no projeto:

- substituição do resolvedor SAT zChaff pelo MiniSat
- integração do código fonte de todos os softwares adicionais envolvidos ao dos softwares do projeto

Com as modificações propostas os softwares adicionais se tornarão apenas módulos dos softwares do projeto, fazendo parte do mesmo executável. Assim a interface e troca de dados entre esses módulos farão parte do software principal, ocorrendo em memória primária ao invés de secundária. Além disso, como esses módulos serão compilados e instanciados junto com os softwares do projeto, não haverá gasto extra de tempo para instanciação de novos processos.

Além da melhora no desempenho do softwares do projeto, a substituição do zChaff pelo MiniSat também implica que o pacote de softwares poderá ser distribuído completo, sob a licença GPLv3.

5 Trabalho Realizado

São listadas nessa sessão as partes relevantes da implementação original dos resolvedores e em seguida as modificações feitas.

As soluções descritas na seção 3 são implementadas nos seguintes softwares, chamados resolvedores PSAT: PSATtoSAT para a redução canônica; PsatColGen para a solução com o método de geração de colunas baseada em resolvedor SAT; e PSATtoMaxSat para a solução com o método de geração de colunas baseada em resolvedor Max-SAT Ponderado.

5.1 Implementação original

Todas a leituras e escritas de arquivos mencionadas a seguir implicam em acesso ao disco rígido.

5.1.1 Chamadas ao zChaff

Chamadas ao zChaff são feitas nos três resolvedores. Para tais chamadas é necessário antes armazenar a instância SAT que desejamos resolver em um arquivo e então chamar o zChaff externamente (executável separado) passando esse arquivo como entrada, que por sua vez escreve sua resposta em um arquivo de saída. Em seguida o resolvedor em questão deve abrir o arquivo de saída produzido pelo zChaff e deve ler a sua resposta.

Tais chamadas ocorrem:

- uma vez, após a construção, em memória, da instância SAT no PSATtoSAT; a resposta dada pelo zChaff é a mesma que o PSATtoSAT deve dar
- $k + 1$ vezes, para verificar a Γ-consistência das colunas da base inicial no PsatColGen e PSATtoMaxSat
- no máximo $k+1$ vezes, para gerar uma coluna no PsatColGen (procedimento que ocorre um número desconhecido maior do que k vezes)

5.1.2 Chamadas ao LA

Chamadas ao inversor de matrizes LA são feitas apenas nas soluções que usam geração de colunas. Para essas chamadas é necessário antes armazenar a matriz que desejamos inverter em um arquivo e então chamar externamente o LA passando esse arquivo como entrada, que

por sua vez escreve a matriz inversa em um arquivo de saída. Em seguida o resolvedor deve abrir tal arquivo de saída e ler a matriz inversa.

Tais chamadas ocorrem:

- uma vez em cada iteração do algoritmo, para inverter a base corrente e usá-la para montar parte das restrições para a coluna que será gerada, no PsatColGen e no PSATtoMaxSat

5.1.3 Chamadas ao wmaxsatz

Chamadas ao wmaxsatz são feitas apenas na geração de colunas baseadas em Max-SAT Ponderado. Para essas chamadas é necessário antes armazenar a instância Max-SAT Ponderado em um arquivo e então chamar o wmaxsatz externamente passando esse arquivo como entrada, que por sua vez escreve a sua resposta em um arquivo de saída. Em seguida o PSATtoMaxSat deve abrir tal arquivo de saída e ler a sua resposta.

Tais chamadas ocorrem:

- uma vez em cada geração de coluna, no PSATtoMaxSat

5.2 Implementação modificada

É definido agora como *módulo* parte de um software que tem função bem definida, diferente do procedimento principal do software, que tem a característica de auxiliar na execução do procedimento principal (e.g. um módulo resolvedor SAT de um software resolvedor PSAT resolve instâncias SAT auxiliando na solução da instância PSAT).

Foram adicionados os seguintes módulos aos seguintes resolvedores:

- *módulo MiniSat* ao PSATtoSAT, PsatColGen e PSATtoMaxSat; derivado do código fonte do resolvedor SAT MiniSat
- *módulo LA* ao PsatColGen e PSATtoMaxSat; derivado do código fonte do inversor de matrizes LA
- *módulo wmaxsatz* ao PSATtoMaxSat; derivado do código fonte do resolvedor Max-SAT Ponderado wmaxsatz

O zChaff não é mais utilizado, e no lugar de todas as chamadas originais feitas a ele agora são feitas chamadas ao módulo MiniSat de cada resolvedor PSAT. A instância SAT em

questão é passada para o módulo, assim como a resposta do módulo, via memória logo após sua construção, eliminando a necessidade de acesso a disco.

Todas as chamadas externas ao LA e ao wmaxsatz foram substituídas por chamadas ao módulo LA e módulo wmaxsatz respectivamente, de cada resolvedor. Assim como com o módulo MiniSat, os dados de entrada e saída de ambos os módulos agora são passados via memória e não via arquivos.

6 Testes

Os testes realizados para medir e comparar as versões originais e modificadas dos resolvedores são baseados em conjuntos de três tipos de instâncias:

- instâncias satisfazíveis
- instâncias insatisfazíveis
- instâncias aleatórias

Os conjuntos de instâncias satisfazíveis e insatisfazíveis são usados para medir o desempenho específico de cada resolvedor, assim como a confiabilidade das suas respostas (i.e. porcentagem de respostas certas), e verificar como as modificações alteram o seu comportamento.

O conjunto de instâncias aleatórias é usado para medir o desempenho geral de cada resolvedor, assim como para verificar como as modificações os alteram.

Para determinar a *confiabilidade* dos resolvedores é medida a porcentagem de respostas certas, i.e. respostas que condizem com o esperado para os conjuntos de instâncias geradas sabidamente satisfazíveis ou insatisfazíveis. Respostas consideradas erradas são: respostas que não condizem com o esperado ou respostas nas quais houve erro de execução.

Os parâmetros usados nos testes apresentados são:

- k: número de variáveis com probabilidade associada
- n: número de variáveis sem probabilidade associada (*variáveis SAT*)
- m: número de cláusulas 3-SAT geradas (usado no teste de instâncias aleatórias)
- l: número de cláusulas SAT geradas (demais instâncias)

Todos os testes foram executados em um computador Intel i7 3.40 GHz×12 com 24GBytes de memória RAM e sistema operacional Ubuntu 12.04.1.

6.1 Instâncias Satisfazíveis

Esse conjunto de testes é formado por instâncias construídas satisfazíveis. Para as instâncias desse teste o valor de k é variado e o valor de n e l são dependentes de k. Para

cada valor de k foram geradas 50 instâncias e foi medido o seu tempo médio de execução para respostas corretas, assim como o número de acertos.

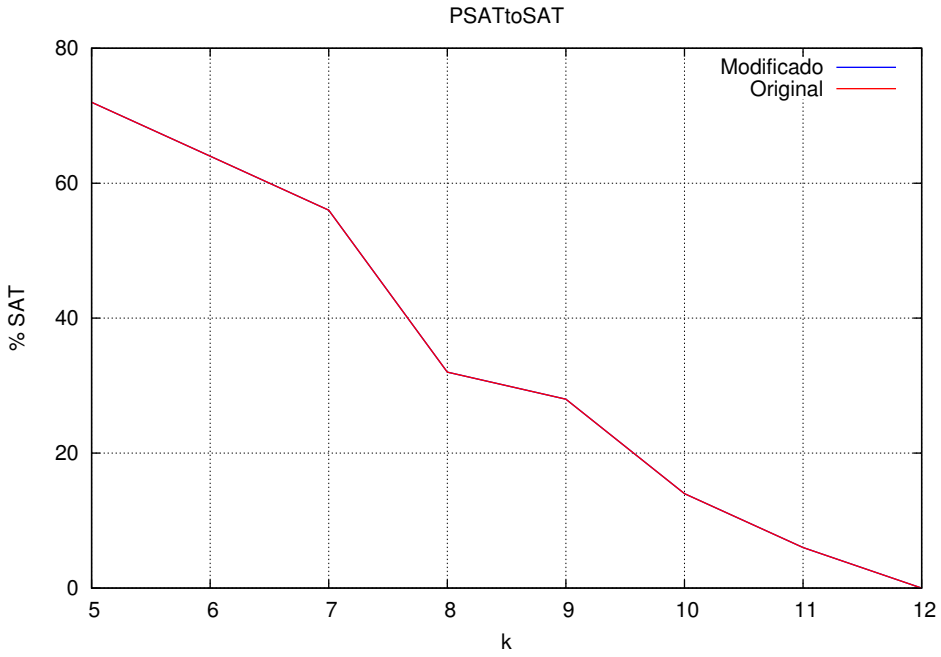

Figura 6.1: confiabilidade das versões do PSATtoSAT para instâncias satisfazíveis

Tanto a versão original quanto a modificada apresentam a mesma confiabilidade (Figura 6.1), indicando que a corretude das suas respostas está ligada a algum fator que não o uso do resolvedor SAT.

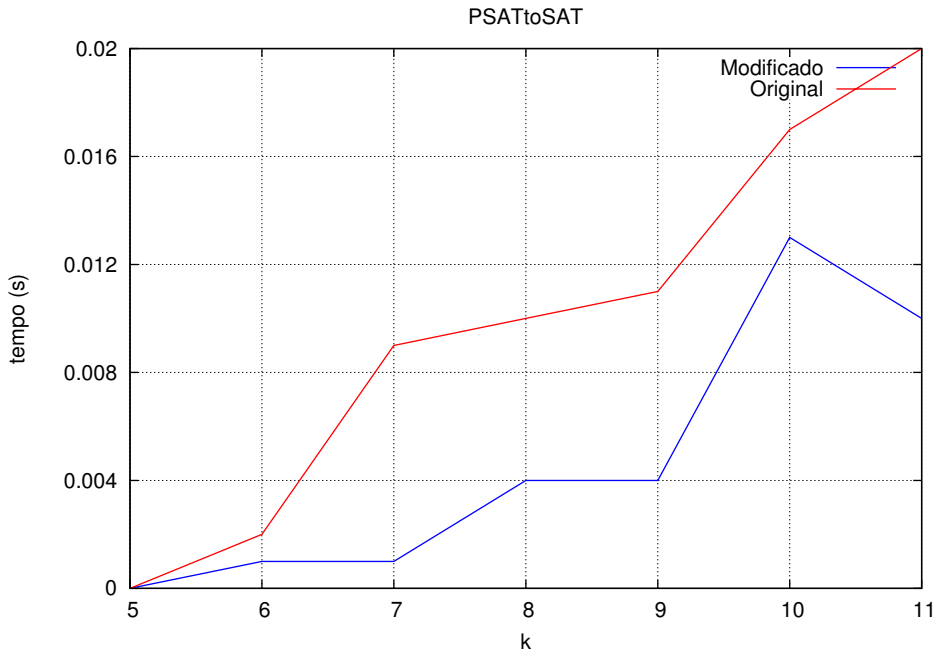

Figura 6.2: desempenho das versões do PSATtoSAT para instâncias satisfazíveis

Mesmo para entradas de tamanho limitado, e com o fato de que o PSATtoSAT faz apenas uma chamada ao resolvedor SAT, é verificado ganho no desempenho (Figura 6.2).

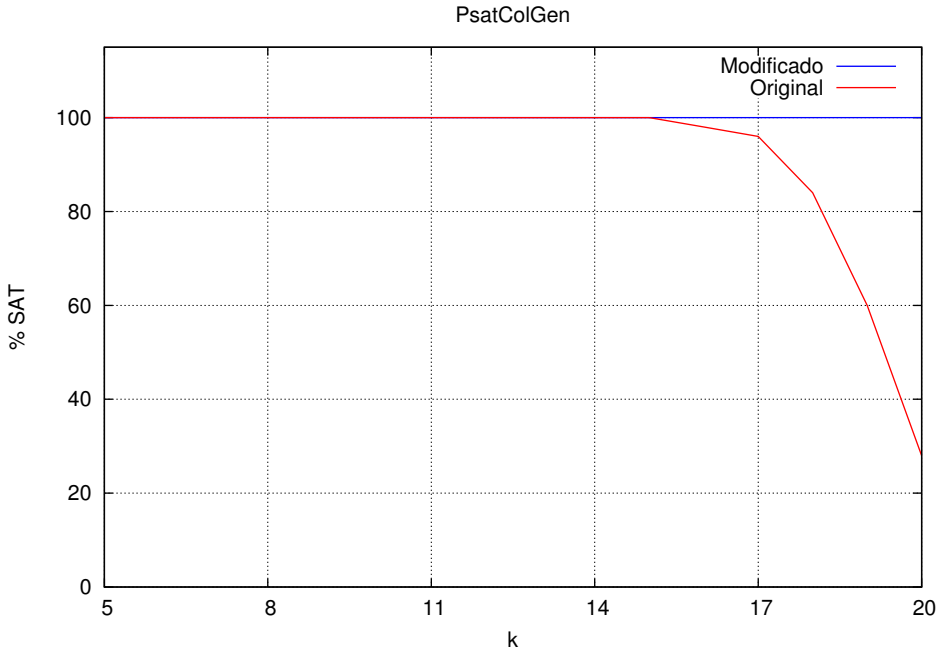

Figura 6.3: confiabilidade das versões do PsatColGen para instâncias satisfazíveis

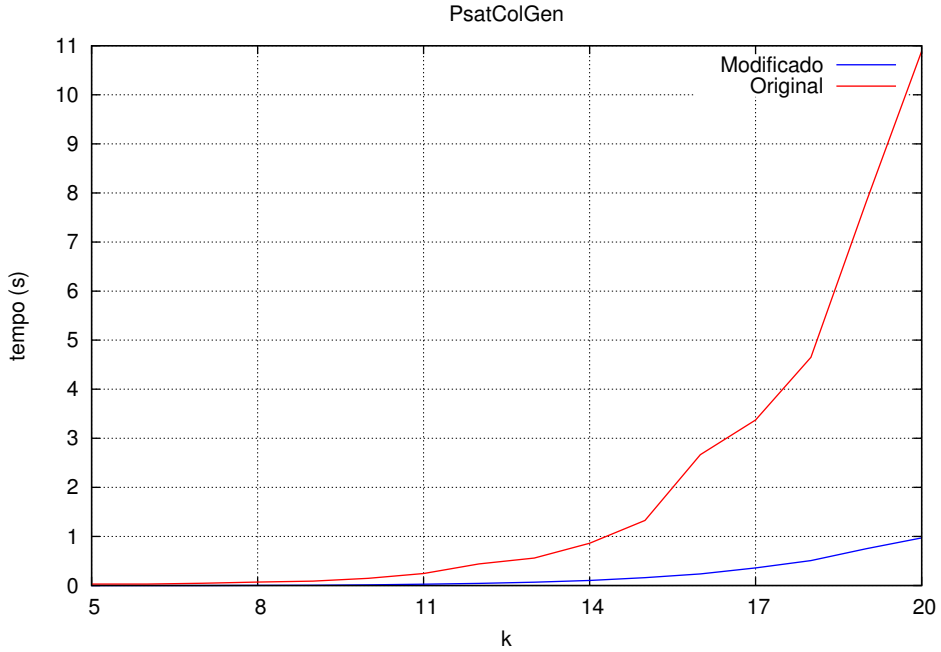

Figura 6.4: desempenho das versões do PsatColGen para instâncias satisfazíveis

A confiabilidade do PsatColGen apresenta melhora com as modificações (Figura 6.3) e é notada uma melhora expressiva no desempenho do resolvedor (Figura 6.4).

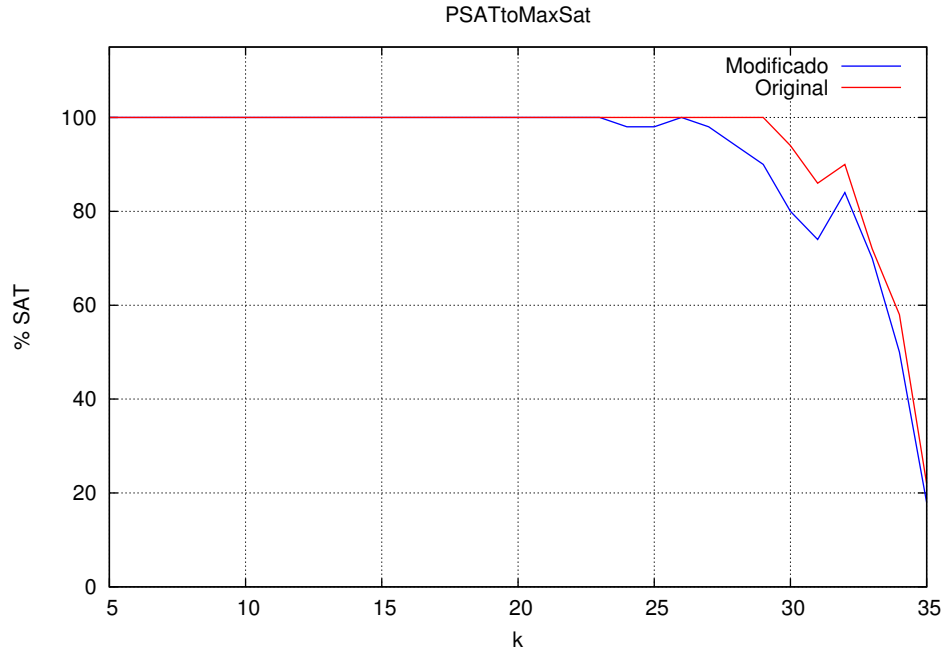

Figura 6.5: confiabilidade das versões do PSATtoMaxSat para instâncias satisfazíveis

Percebemos que ambas as versões do PSATtoMaxSat apresentam confiabilidade semelhante (Figura 6.5).

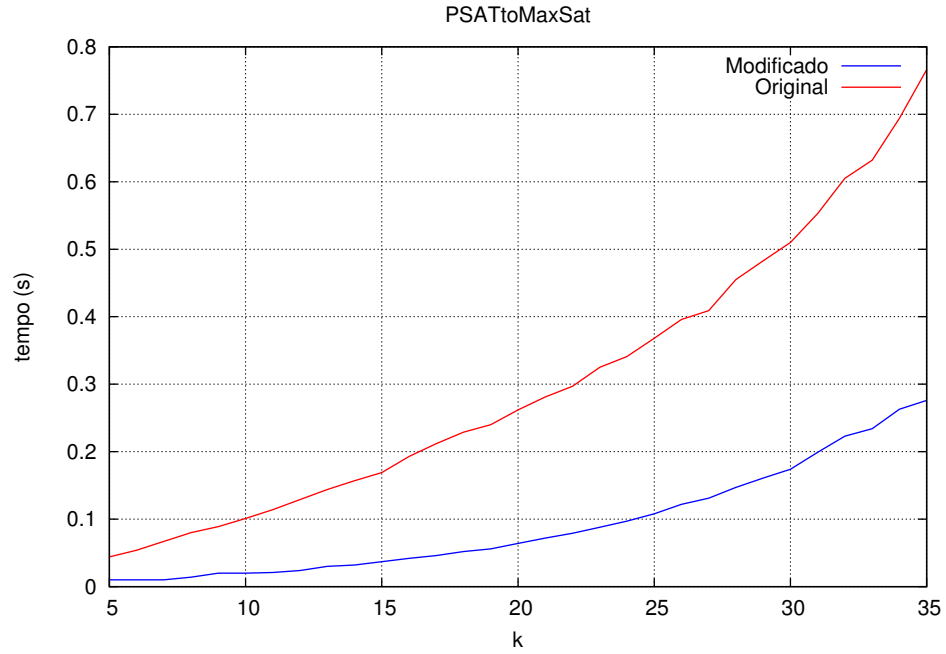

Figura 6.6: desempenho das versões do PSATtoMaxSat para instâncias satisfazíveis

Também é notado um melhor desempenho na versão modificada (Figura 6.6).

6.2 Instâncias Insatisfazíveis

Esse conjunto de testes é formado por instâncias construídas insatisfazíveis. De modo semelhante ao conjunto de instâncias satisfazíveis, o valor de k é variado e o valor de n e l são pseudo-aleatórios dependentes de k. Para cada valor de k foram geradas 50 instâncias e foi medido o seu tempo médio de execução para respostas corretas, assim como o número de acertos.

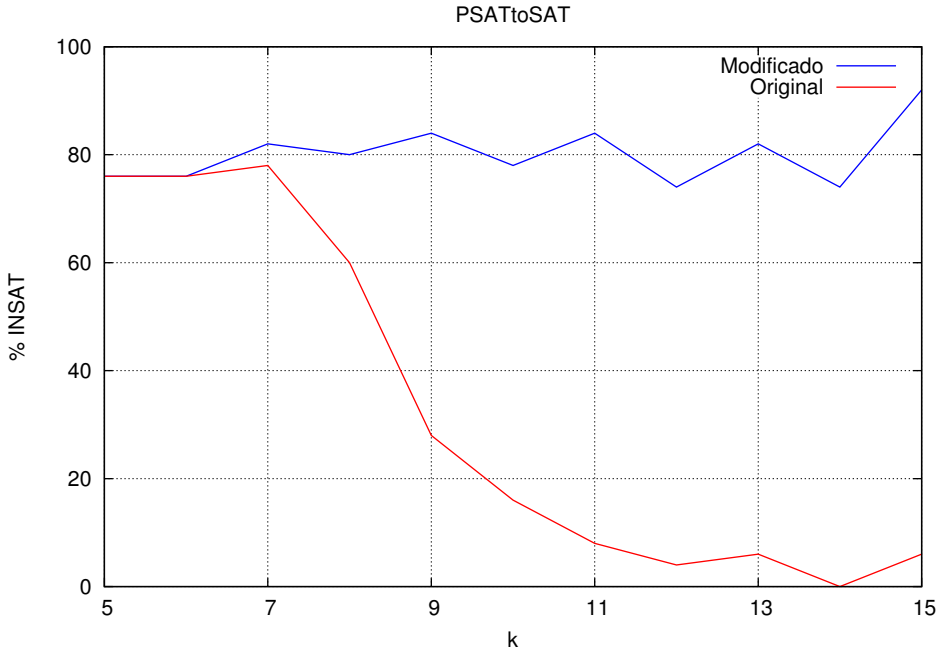

Figura 6.7: confiabilidade das versões do PSATtoSAT para instâncias insatisfazíveis

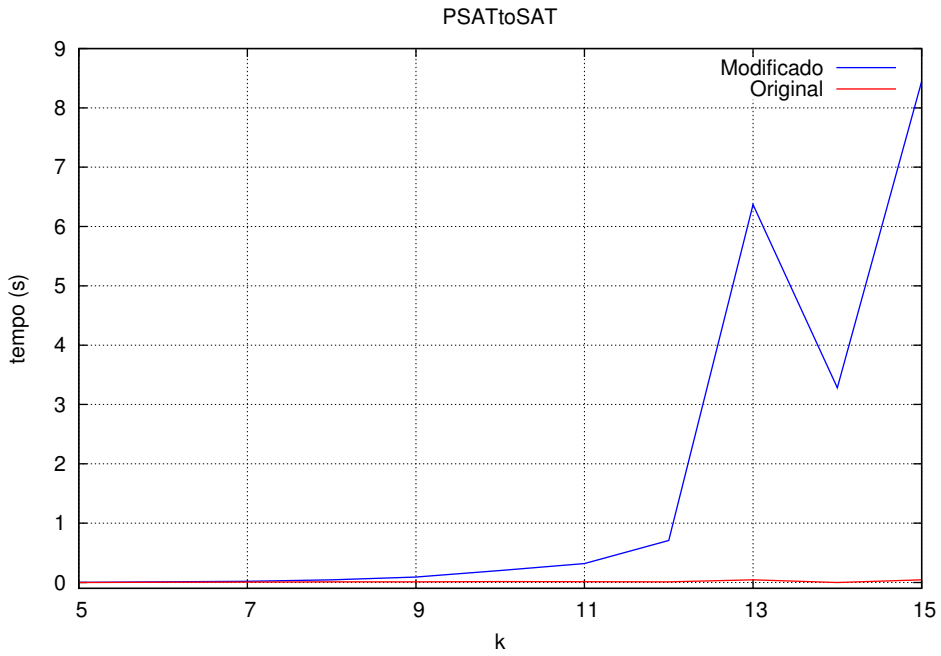

Figura 6.8: desempenho das versões do PSATtoSAT para instâncias insatisfazíveis (curva completa)

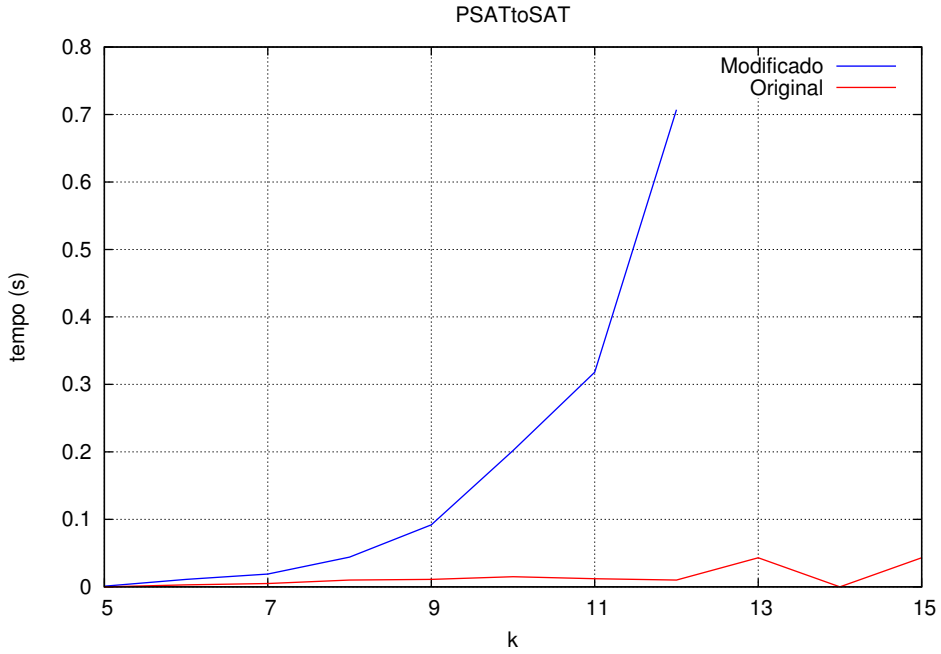

Figura 6.9: desempenho das versões do PSATtoSAT para instâncias insatisfazíveis (curva local)

A diferença de confiabilidade expressiva entre as versões do PSATtoSAT (Figura 6.7) pode indicar que a aparente melhor performance da versão original seja resultado de que apenas instâncias fáceis (facilmente decidíveis) foram avaliadas corretamente por ela. Percebemos um aumento na diferença do tempo de execução a partir de $k = 8$, onde a confiabilidade é de 60% e continua a diminuir.

Ao mesmo tempo, verificamos que a versão modificada apresenta confiabilidade elevada (Figuras 6.8 e 6.9).

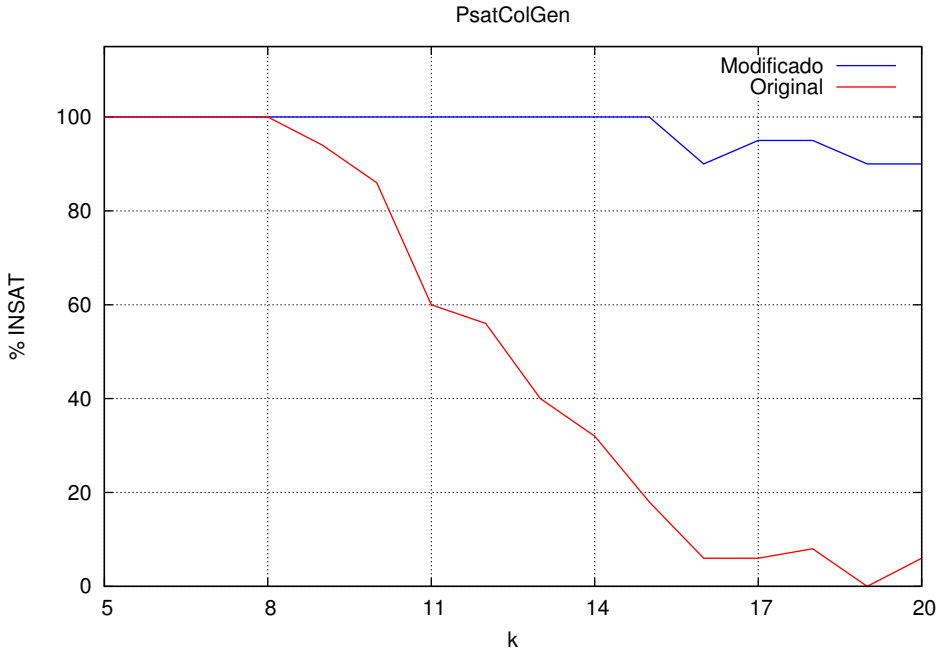

Figura 6.10: confiabilidade das versões do PsatColGen para instâncias insatisfazíveis

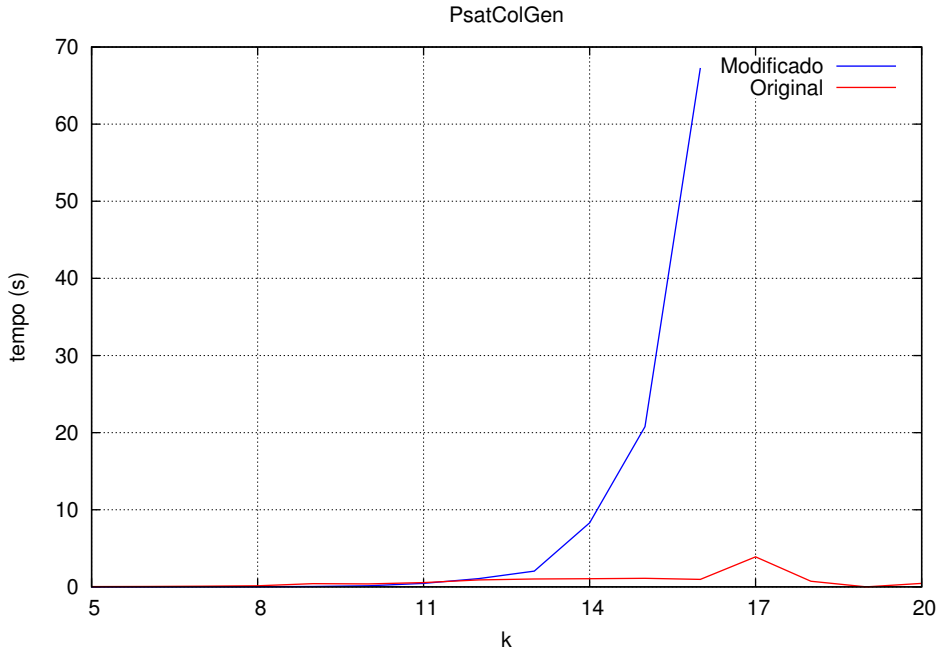

Figura 6.11: desempenho das versões do PsatColGen para instâncias insatisfazíveis (curva completa)

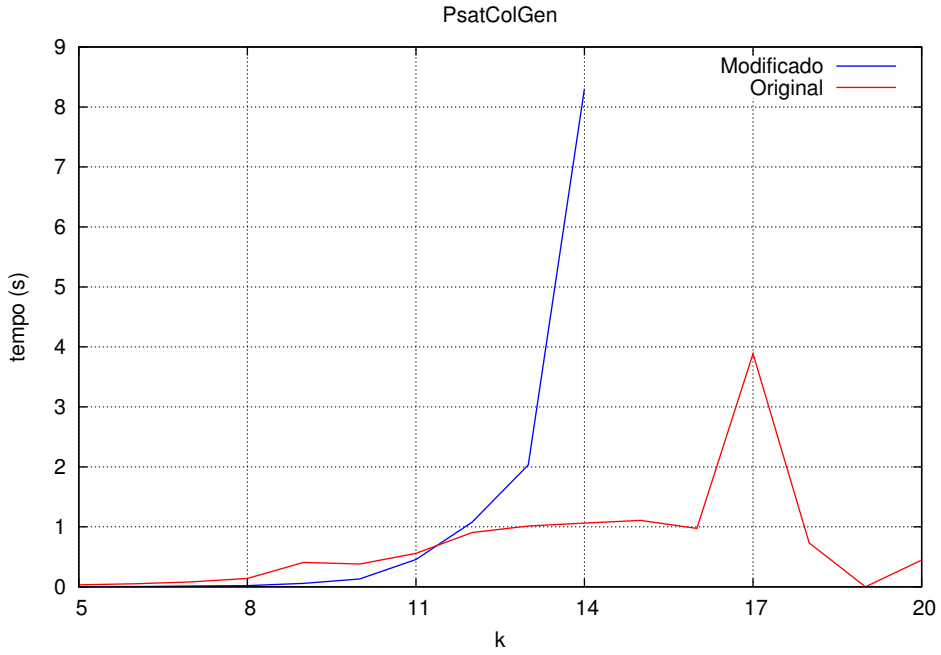

Figura 6.12: desempenho das versões do PsatColGen para instâncias insatisfazíveis (curva local)

Semelhante ao PSATtoSAT, é possível que a diferença de confiabilidade das duas versões do PsatColGen (Figura 6.10) indique que a performance elevada da versão original venha do fato de que apenas instâncias fáceis tenham sido corretamente avalidas, sendo decididas mais rapidamente. Percebemos que a disparidade nos tempos de execução fica expressiva a partir $k \approx 12$, onde a confiabilidade é menos de 60% e continua a diminuir.

Também, assim como com o PSATtoSAT, podemos ver que a versão modificada apresenta maior confiabilidade (Figuras 6.11 e 6.12).

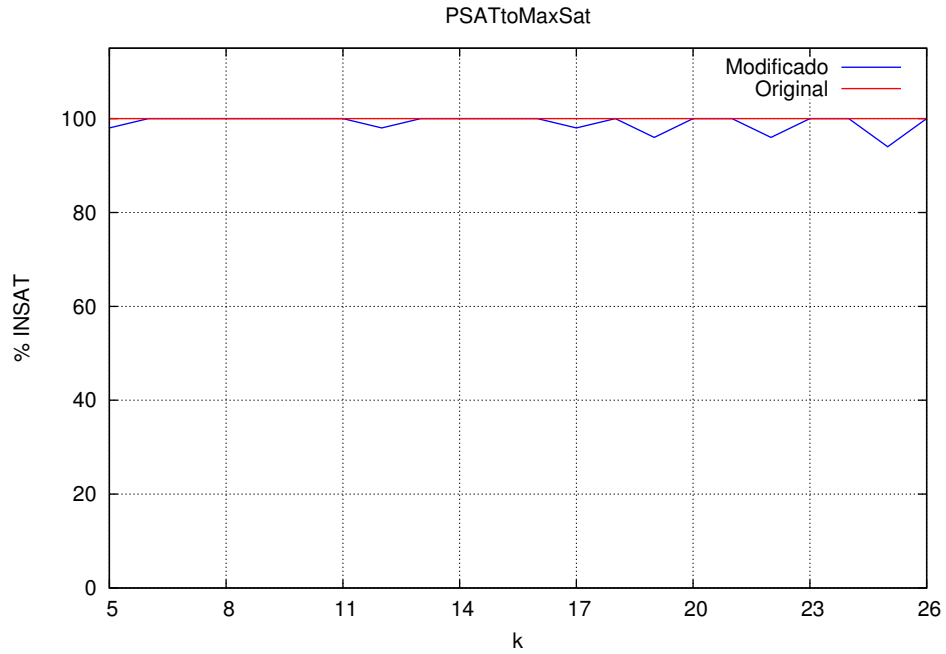

Figura 6.13: desempenho das versões do PSATtoMaxSat para instâncias insatisfazíveis

Notamos que a confiabilidade de ambas as versões do PSATtoMaxSat mantém-se elevada (Figura 6.13).

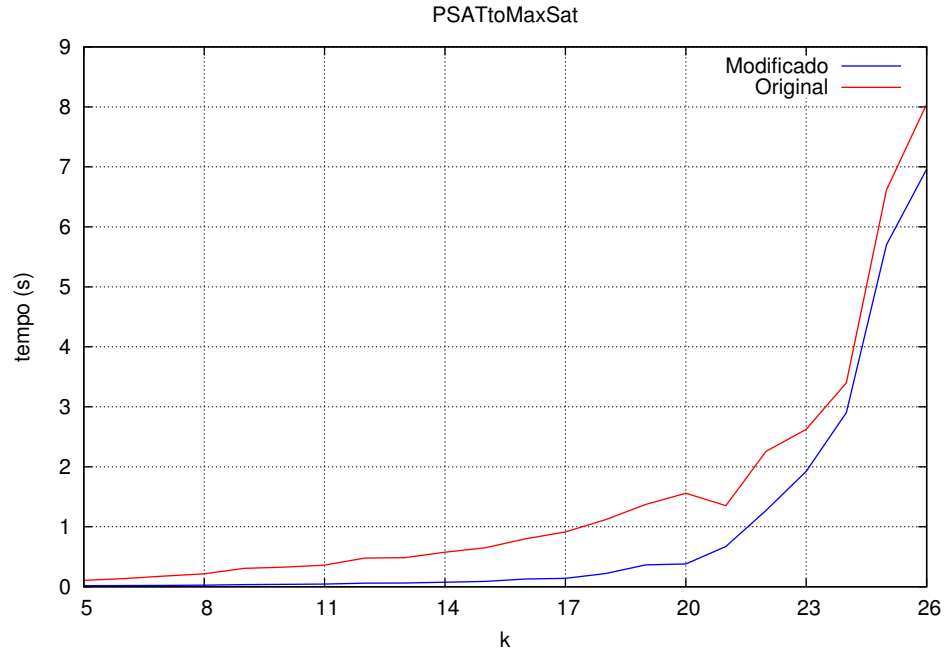

Figura 6.14: desempenho das versões do PSATtoMaxSat para instâncias insatisfazíveis

Também podemos notar que há melhora no desempenho do resolvedor (Figura 6.14).

6.3 Instâncias Aleatórias

Esse conjunto de testes é constituído por instâncias com k variáveis probabilísticas, n variáveis SAT e m cláusulas 3-SAT. Para cada conjunto de testes k e n são mantidos fixos e m é variado de 0 a $5n$. Para cada valor de m foram geradas 50 instâncias aleatoriamente e foi medido o seu tempo médio de execução. Além do tempo de execução também foi medida a porcentagem de instâncias satisfazíveis, para cada valor de m (% SAT).

Para esse conjunto de testes não foi medida a confiabilidade dos resolvedores devido a dois fatos: não sabemos, como nos outros testes, a satisfatibilidade das instâncias de antemão; e o número de respostas diferentes entre a versão modificada e original de cada resolvedor foi muito baixo.

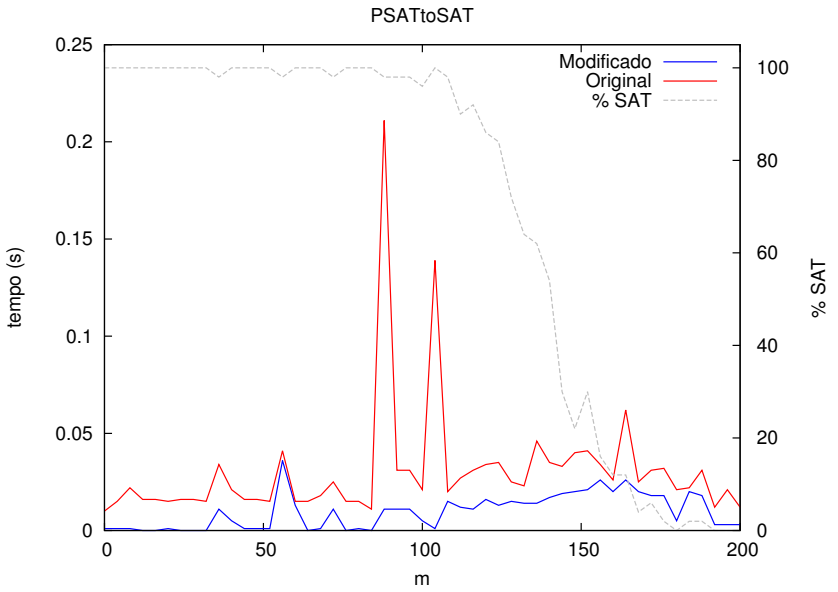

Figura 6.15: $n = 40, k = 4$

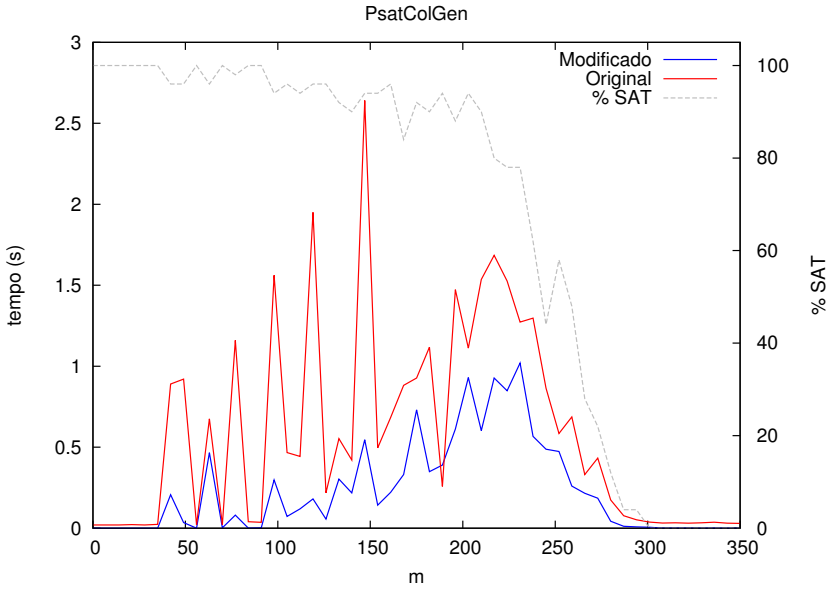

Figura 6.16: $n = 80, k = 12$

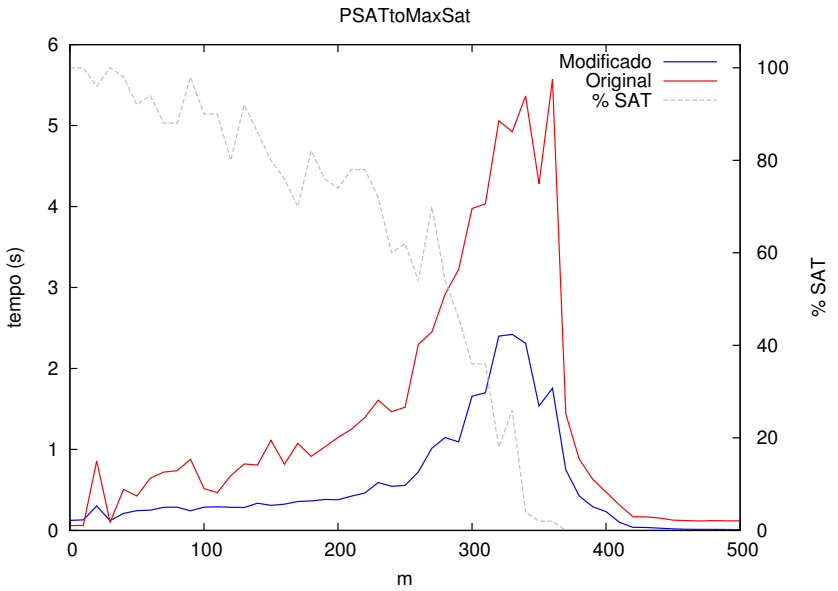

Figura 6.17: $n = 100, k = 20$

Para os três resolvedores verificamos uma melhora significativa no seu desempenho (Figuras 6.15, 6.16 e 6.17) em todos os casos: instâncias principalmente satisfazíveis (região esquerda dos gráficos); principalmente insatisfazíveis (região direita); instâncias mistas (transição de satisfazíveis para insatisfazíveis).

7 Conclusão

Pudemos perceber que para as instâncias satisfazíveis e aleatórias as modificações feitas resultaram em um ganho significativo no desempenho dos três resolvedores. Onde tal ganho foi mais expressivo com o PsatColGen e com o PSATtoMaxSat, devido a possuírem maior limiar de uso, no tamanho da entrada, mas principalmente pelo fato de fazerem uso de softwares externos diversas vezes em uma mesma execução, o que antes resultava em diversos acessos a disco além do tempo de execução do software em si.

Para instâncias insatisfazíveis o desempenho medido a princípio não está de acordo com o esperado em dois dos três resolvedores. Seria esperado que a versão modificada consumisse menos tempo, mas é verificado o contrário. Quando analisamos a diferença de confiabilidade das versões percebemos que esse aparente pior desempenho da versão modificada pode ser resultado do fato dela responder corretamente para um número bem maior de instâncias. Instâncias as quais possivelmente apresentam maior dificuldade de decisão, e por sua vez consomem mais tempo de execução.

Sobre a confiabilidade das respostas para instâncias satisfazíveis devemos analisar cada resolvedor separadamente.

Para o PSATtoSAT a confiabilidade de ambas as versões diminui igualmente. Isso mostra que essa perda de confiabilidade não é decorrente do resolvedor SAT empregado, mas sim da implementação do resolvedor.

Para o PsatColGen a confiabilidade da versão modificada mantém-se elevada enquanto da versão original diminui. Assim percebemos que, se a utilização das técnicas empregadas nesse resolvedor fazem com que ele passe a responsabilidade de decisão sobre o problema para o resolvedor SAT, então a confiabilidade do resolvedor MiniSat usado na versão modificada é maior que do resolvedor zChaff na versão original.

Por fim, para o PSATtoMaxSat a confiabilidade mantém-se semelhante na versão modificada, mostrando que a diferença principal entre as versões é apenas o desempenho.

II

Parte Subjetiva

1 Agradecimentos

O trabalho de formatura supervisionado começa em meados de abril, concluindo-se em dezembro. Quando fui pedir orientação, o professor Marcelo Finger já avisou que a partir do segundo semestre desse ano, estaria fora do país. Dessa forma, a orientação que recebi desde agosto tem sido remota. O que a princípio poderia parecer um grande empecilho acabou por ser não mais do que um detalhe no processo.

Assim gostaria de agradecer a atenção que recebi do professor Marcelo Finger que não deixou de forma alguma a distância impedir uma boa orientação.

Gostaria de agradecer também a Glauber de Bona, que foi de indispensável ajuda para o entendimento e desenvolvimento desse trabalho.

2 Desafios e frustrações

O maior desafio de todos foi lidar com código de outras pessoas. Os softwares envolvidos no projeto são de alta complexidade e não estão idealmente documentados. Juntando-se isso ao fato de que só um deles foi de fato desenvolvido com o intuito de ser reutilizado (MiniSat), a tarefa de integrar o código de todos foi muito trabalhosa e desgastante.

Outro desafio também foi ter que elaborar rotinas de testes que fossem automáticas e flexíveis ao mesmo tempo. O resultado foi uma combinação de testes realizados em um executável testador feito em C++, que executava os testes com instâncias de nome padronizado, coletava o resultado e montava uma tabela; script que executava o testador para tipos de instâncias diferentes e intervalos de instâncias diferentes (querer executar *todas* a instâncias para um resolvedor antes de começar o próximo resolvedor significava em ter que esperar muito tempo para ver um resultado e possivelmente identificar um erro, então os resolvedores foram testados em intervalos menores); e por fim um conjunto de scripts para montagem dos gráficos.

Nunca tendo tido que fazer baterias de testes desse jeito antes, foi um desafio interessante que resultou em um método razoavelmente simples e flexível para realização de testes comparativos.

3 Relação com matérias do curso

Matérias do curso que tiveram a maior relação com o trabalho desenvolvido:

MAC0338 Análise de Algoritmos
MAC0315 Programação Linear
MAC0239 Métodos Formais em Programação

Certamente as matérias essenciais que me deram os conceitos necessários para entender todo o conteúdo envolvido nesse trabalho. Desde o conceito de classes de complexidade e a importância do SAT mostrados em Análise de Algoritmos, até princípios de lógica em Métodos Formais e praticamente todo o conteúdo de Programação Linear, que são a base das soluções nas quais esse trabalho foi desenvolvido.

MAC0110 Introdução à Computação
MAC0122 Princípios de Desenvolvimento de Algoritmos
MAC0211 Laboratório de Programação I
MAC0323 Estruturas de Dados

Matérias sem as quais não teria desenvolvido minha habilidade para programar, muito menos para realizar as tarefas necessárias nesse trabalho.

Referências

[1] Boole, G. 1854. In *An Investigation on the Laws of Thought*, Macmillan, London. Available on project Gutenberg at `http://www.gutenberg.org/etext/15114`.

[2] Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing (STOC)*, p. 151–158.

[3] Eén, N. and Sörensson, N. 2003. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003*, Santa Margherita Ligure, Italy, p. 502-518.

[4] Finger, M. and de Bona, G. 2011. Probabilistic Satisfiability: Logic-Based Algorithms and Phase Transition. In Toby Walsh, NICTA and University of NSW *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, AAAI Press/International Joint Conferences on Artificial Intelligence, Barcelona, p. 528-533.

[5] Finger, M. and de Bona, G. 2012. Probabilistic Satisfiability: Algorithms with the presence and absence of a phase transition *Manuscript in preparation.*

[6] Georgakopoulos, G., Kavvadias, D. and Papadimitriou, C. H. 1988. Probabilistic satisfiability. In *Journal of Complexity, 4(1)*, p. 1-11.

[7] Hailperin, T. 1984. Probability Logic. In *Notre Dame Journal of Formal Logic, 25(3)*, p. 198–212.

[8] Hansen, P. and Jaumard, B. 2000. Probabilistic satisfiability. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems. Vol.5*, Springer, Netherlands, p. 321.

[9] Li, C. M., Manya, F., Mohamedou, N. and Planes, J. . Exploiting Cycle Structures in Max-SAT. In *Proceedings of 12th International Conference on the Theory and Applications of Satifiability Testing (SAT)*, Springer, LNCS 5584, Swansea, United Kingdom, p. 467-480.

[10] Malik, S. and Zhang, L. . Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM 52*, p. 76-82.

[11] Marquies-Silva, J. 2008. Practical Applications of Boolean Satisfiability. In *Workshop on Discrete Event Systems (WODES'08)*, Goteborg, p. 74-80.

[12] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L. and Malik, S. 2001. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference (DAC '01)*, ACM, New York, NY, USA, p. 530-535.

[13] Nilsson, N. 1986. Probabilistic logic. In *Artificial Intelligence, 28(1)*, p. 71–87.