BRUNO CÉSAR RIBAS

UM MÉTODO DE PRÉ-PROCESSAMENTO DE FÓRMULAS SAT E PSEUDO-BOOLEAN BASEADO EM TÉCNICAS DE PROGRAMAÇÃO LINEAR INTEIRA MISTA

CURITIBA

2015

BRUNO CÉSAR RIBAS

UM MÉTODO DE PRÉ-PROCESSAMENTO DE FÓRMULAS SAT E PSEUDO-BOOLEAN BASEADO EM TÉCNICAS DE PROGRAMAÇÃO LINEAR INTEIRA MISTA

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Fabiano Silva

CURITIBA

2015

R482m Ribas, Bruno César
Um método de pré-processamento de fórmulas SAT e pseudo-boolean baseado em técnicas de programação linear inteira mista/ Bruno César Ribas. – Curitiba, 2015.
143 f. : il. color. ; 30 cm.

Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2015.

Orientador: Fabiano Silva .
Bibliografia: p. 134-141.

1. Complexidade computacional. 2. Otimização matemática. 3. Programação linear. I. Universidade Federal do Paraná. II.Silva, Fabiano. III. Título.

CDD: 511.352

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Comissão Examinadora da defesa do(a) aluno(a) de Doutorado em Ciência da Computação, Bruno César Ribas , avaliamos a de tese de doutorado intitulado "UM MÉTODO DE PRÉ-PROCESSAMENTO DE FÓRMULAS SAT E PSEUDO-BOOLEAN BASEADO EM TÉCNICAS DE PROGRAMAÇÃO LINEAR INTEIRA MISTA", cuja defesa pública, foi realizada no dia 21 de setembro de 2015. Após avaliação, decidimos pela:

(X) **Aprovação** do(a) candidato(a). ( ) **Reprovação** do(a) candidato(a).

Curitiba, 21 de setembro de 2015.

Prof. Dr. Fabiano Silva
PPGINF/UFPR – Orientador

Prof. Dr. Marcelo Finger
IME/USP – Membro Externo

Prof. Dr. Adolfo Gustavo Serra-Seca-Neto
UTFPR – Membro Externo

Prof. Dr. Alexandre Ibrahim Direne
PPGINF – Membro Interno

Prof. Dr. Renato Carmo
PPGINF – Membro Interno

AGRADECIMENTOS

Agradeço à minha família, meus amigos, professores e funcionários do DINF-UFPR pelo apoio desde a graduação até a conclusão do Doutorado.

SUMÁRIO

LISTA DE FIGURAS xii

LISTA DE TABELAS xiv

RESUMO xv

ABSTRACT xvi

1 INTRODUÇÃO 1

1.1 Contribuição 3

1.2 Organização do Texto 4

2 TÉCNICAS PARA RESOLVEDORES SAT, PSEUDO-BOOLEAN E PROGRAMAÇÃO INTEIRA MISTA 6

2.1 Introdução à Lógica Proposicional 6

2.2 Satisfatibilidade 7

2.2.1 Forma Normal Conjuntiva 9

2.2.1.1 Conversão para CNF 9

2.2.2 O algoritmo DPLL 10

2.2.3 Heurísticas para ramificação 14

2.2.4 BCP rápido 16

2.2.5 Aprendizado de cláusulas e retrocesso não-cronológico 17

2.2.6 Outras otimizações 19

2.3 Programação Inteira Mista 21

2.3.1 Branch-and-Bound 21

2.3.2 Branch-and-Cut 23

2.4 Satisfatibilidade Pseudo-Booleana 24

2.4.1 Representação Pseudo-Booleana 24

2.4.2 Propagação . . . 28

2.4.3 Aprendizado . . . 31

2.5 Considerações finais . . . 32

3 REVISÃO DOS MÉTODOS DE PRÉ-PROCESSAMENTO 34

3.1 Pré-processamento para SAT . . . 35

3.2 Pré-processamento para Programação Inteira Mista . . . 37

3.2.1 Técnicas Básicas de Pré-processamento . . . 38

3.2.2 Técnicas de Sondagem Básica . . . 39

3.2.3 Implicações Lógicas . . . 41

3.2.4 Técnica Aprimorada de Sondagem . . . 43

3.2.5 Desigualdades Clique . . . 45

3.3 Fortalecimento Pseudo-Booleano . . . 46

3.4 Considerações Finais . . . 49

4 MODELO PROPOSTO 50

4.1 Sondagem . . . 51

4.2 Aprendizado . . . 55

4.2.1 Aprendizado Avançado . . . 58

4.3 Fortalecimento . . . 62

4.4 Subjugação de Restrições . . . 63

4.5 Considerações Finais . . . 67

5 AVALIAÇÃO EXPERIMENTAL 69

5.1 Teste da Sondagem . . . 70

5.1.1 Fórmulas CNF . . . 71

5.1.2 Fórmulas Pseudo-Booleanas . . . 81

5.2 Teste de Aprendizado . . . 86

5.2.1 Fórmulas CNF . . . 87

5.2.2 Fórmulas pseudo-Booleanas . . . 90

5.3 Teste do Fortalecimento . . . 96

5.3.1 Fórmulas CNF 98

5.3.2 Fórmulas pseudo-Boolean 100

5.4 pBCR completo 106

5.4.1 Fórmulas CNF 107

5.4.2 Fórmulas pseudo-Boolean 109

5.4.3 Mudanças nas fórmulas 110

5.5 Considerações Finais 115

6 PSEUDO-BOOLEAN APLICADO À CONSOLIDAÇÃO DE MÁQUINAS VIRTUAIS 118

6.1 A primeira versão 119

6.2 PBFVMC 121

6.2.1 O modelo proposto 121

6.2.2 Discussão do PBFVMC 123

6.3 Avaliação Experimental 124

6.4 Considerações Finais 126

7 CONCLUSÃO 130

BIBLIOGRAFIA 134

APÊNDICES 142

A ANÁLISE DE SIGNIFICÂNCIA ESTATÍSTICA 142

LISTA DE FIGURAS

2.1 Diagrama de uma fórmula CNF. 9

2.2 Grafo de decisão com conflito na variável x_3 20

3.1 Grafo auxiliar para a formação da desigualdade clique 47

4.1 Diagrama do processo de aplicação do pré-processador *pBCR* em uma fórmula de entrada 51

4.2 Etapas de construção do grafo de Aprendizado Avançado, para o exemplo . 61

5.1 Diagrama do processo de aplicação do pré-processador *pBCR* no experimento da Sondagem, etapa 1 71

5.2 Comparação estatística do pós-teste de Nemenyi 73

5.3 Comparação dos tempos de execução do LINGELING nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do LINGELING no eixo y contra o LINGELING com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência. 75

5.4 Comparação dos tempos de execução do CLASP nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência. 77

5.5 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência. 78

5.6 Comparação dos tempos de execução do GLUCOSE nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do GLUCOSE no eixo y contra o GLUCOSE com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência. 79

5.7 Comparação estatística do pós-teste de Nemenyi com os resolvedores LINGELING, CLASP e GLUCOSE em cima das fórmulas originais e pré-processadas pelo *pBCR* executando apenas a etapa 1, sondagem, sem considerar o tempo de processamento do *pBCR* 81

5.8 Comparação estatística do pós-teste de Nemenyi com os resolvedores LINGELING, CLASP e GLUCOSE em cima das fórmulas originais e pré-processadas pelo *pBCR* executando apenas a etapa 1, sondagem, considerando o tempo de processamento do *pBCR* 82

5.9 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência. 83

5.10 Comparação dos tempos de execução do CLASP nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência. 85

5.11 Comparação dos tempos de execução do SCIP nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP no eixo y contra o SCIP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência. 87

5.12 Diagrama do processo de aplicação do pré-processador *pBCR* no experimento da Aprendizado, etapa 2 88

5.13 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, do benchmark CNF, originais e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas com as restrições aprendidas no eixo x. A linha $f(x) = x$ é colocada como referência. 89

5.14 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, do benchmark CNF, sondadas e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas sondadas no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência. 90

5.15 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, originais e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência. 91

5.16 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, sondadas e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas sondadas no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência. 92

5.17 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, originais e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência. 93

5.18 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, sondadas e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas sondadas no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência. 94

5.19 Comparação dos tempos de execução do SCIP nas fórmulas pseudo-Booleanas, originais e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP com as fórmulas originais no eixo y contra o SCIP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência. 95

5.20 Comparação dos tempos de execução do SCIP nas fórmulas pseudo-Booleanas, sondadas e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP com as fórmulas sondadas no eixo y contra o SCIP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência. 96

5.21 Diagrama do processo de aplicação do pré-processador *pBCR* no experimento do Fortalecimento, etapa 3 97

5.22 Comparação dos tempos de execução do CLASP nas fórmulas CNF originais e com fortalecimento de crescimento 2 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 99

5.23 Comparação dos tempos de execução do CLASP nas fórmulas CNF originais e com fortalecimento de crescimento 4 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 100

5.24 Comparação dos tempos de execução do CLASP nas fórmulas CNF originais e com fortalecimento de crescimento 8 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 101

5.25 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 2 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 102

5.26 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 4 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 103

5.27 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 8 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 104

5.28 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 2 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 105

5.29 Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 4 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 106

5.30 Comparação dos tempos de execução do SCIP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 2 pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 107

5.31 Comparação dos tempos de execução do SCIP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 4 pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência. 108

5.32 Restrições eliminadas nas 549 fórmulas em CNF nas configurações de sondagem, fortalecimento de crescrimento 2, 4 e 8 do *pBCR* 111

5.33 Variáveis eliminadas nas 549 fórmulas em CNF nas configurações de sondagem, fortalecimento de crescrimento 2, 4 e 8 do *pBCR* 112

5.34 Restrições eliminadas nas 114 fórmulas pseudo-Booleanas nas configurações de sondagem, fortalecimento de crescimento 2, 4 e 8 do *pBCR* 113

5.35 Variáveis eliminadas nas 114 fórmulas pseudo-Booleanas nas configurações de sondagem, fortalecimento de crescimento 2, 4 e 8 do *pBCR* 114

5.36 Restrições eliminadas nas 650 fórmulas pseudo-Booleanas de otimização nas configurações de sondagem, fortalecimento de crescimento 2 e 4 do *pBCR*115

5.37 Variáveis eliminadas nas 650 fórmulas pseudo-Booleanas de otimização nas configurações de sondagem, fortalecimento de crescimento 2 e 4 do *pBCR* . 116

LISTA DE TABELAS

5.1 Classificação Média dos resolvedores no tempo utilizado para decidir que a fórmula em questão é insatisfatível. 73

5.2 Tempo de execução em segundos do resolvedor CLASP nas fórmulas sondadas que foram resolvidas mas que tiveram o tempo limite excedido em suas configurações originais. 76

5.3 Classificação média dos resolvedores LINGELING, CLASP, GLUCOSE nas fórmulas originais e sondadas sem considerar o tempo de processamento do *pBCR* 81

5.4 Classificação média dos resolvedores LINGELING, CLASP, GLUCOSE nas fórmulas originais e sondadas considerando o tempo de processamento do *pBCR* 82

5.5 Quantidade de fórmulas resolvidas pelos resolvedores GLUCOSE, LINGELING, CLASP e CLASP em pseudo-Boolean dentre as 549 fórmulas CNF . 108

5.6 Quantidade de fórmulas resolvidas pelo CLASP e SCIP dentre as 114 fórmulas pseudo-Booleanas 109

5.7 Quantidade de fórmulas em que o CLASP e o SCIP encontraram a solução ótima dentre as 650 fórmulas de otimização pseudo-Booleanas 110

6.1 Comparação dos tamanhos das fórmulas da primeira formulação e da PBFVMC. A tabela mostra os trabalhos de 25%, 50%, 75%, 85%, 90%, 95%, 98% and 99% para o subconjunto de 32, 64, 128, 256 e 512 *hardware*. 127

6.2 Tempos de execução por instância utilizando o resolvedor SAT4J-PB para a primeira formulação e PBFVMC. O tempo limite foi definido em 14400s e TLE representa que o tempo limite foi alcançado. 128

6.3 Tempo de execução com o resolvedor SAT4J-PB para encontrar alguma valoração satisfatível para a fórmula. 129

A.1 Valores de $q_{0.05}$ para diferentes valores de K no pós-teste de Nemenyi. . . . 143

RESUMO

Ao longo da última década, resolvedores de Satisfatibilidade Booleana (SAT) e Programação Inteira Linear (ILP) melhoraram significativamente com a introdução de novos algoritmos, que permitiram o tratamento de um conjunto mais abrangente de problemas desafiadores e do mundo real, a exemplo, planejamento [39] e verificação de microprocessadores [66]. Existe, também, a representação pseudo-Booleana dentro do escopo de SAT, onde cada variável possui um coeficiente inteiro associado, e as restrições passam a ser desigualdades. A representação pseudo-Booleana é bastante rica e é possível modelar problemas reais facilmente utilizando essa abordagem. Apesar dessa representação ser ligeiramente diferente das representações tradicionais de SAT, ficando mais próxima da representação ILP, as técnicas utilizadas para resolver os problemas são bastante intercambiáveis, possibilitando o aproveitamento dos avanços de SAT nas últimas décadas.

Resolvedores tradicionais falham em resolver diversos problemas por sua demanda por tempo de processamento, e por isso as comunidades têm buscado maneiras de pré-processar as fórmulas [25, 58, 50, 5] a fim de gerarem uma nova fórmula equisatisfatível e que demande menos tempo de processamento para serem resolvidas.

Nesta tese, propomos e implementamos um método de pré-processamento de fórmulas pseudo-Booleanas, *pseudo-Boolean Constraint Reduction - pBCR*, utilizando técnicas da comunidade de pesquisa operacional no intuito de aplicar em fórmulas de diversos domínios de interesse da comunidade de SAT e pseudo-Boolean. Mostramos que a aplicação destas técnicas possui um tempo viável de processamento. Resultados experimentais foram feitos e analisados, permitindo mostrar que o tempo de pré-processamento mantém o tempo total, resolver a fórmula e pré-processar, viável na maioria dos casos.

Aplicamos o método proposto sobre o problema da consolidação de máquinas virtuais. As fórmulas geradas por este estudo de caso foram usadas como parte do domínio de testes da competição de resolvedores pseudo-Booleanos de 2015.

ABSTRACT

Over the last decade, Boolean Satisfiability (SAT) solver and Integer Linear Programming (ILP) have improved significantly with the introduction of new intelligent algorithms which allowed to solve more challenging and broader range of problems, for instance, planning [39] and microprocessor verification [66]. Inside the SAT scope, there is the pseudo-boolean representation, where each variable has an integer coefficient and restrictions becomes inequalities. The pseudo-Boolean representation is very rich and it is possible model real world problems easily using this approach. Although this representation is slightly different from traditional representations of SAT, being closer to ILP, the techniques used to solve problems are very interchangeable, enabling the possibility to use recent advances from SAT of the last decade.

Traditional solvers fail to solve several problems for their processing demand, and both communities are in pursue of ways to pre-process the formulae [25, 58, 50, 5] in order to generate a new equisatisfiable formula that demands less processing time to be solved.

A new pre-processor of pseudo-Boolean formulae has been implemented, named *pseudo-Boolean Constraint Reduction - pBCR*, using techniques from the operations research community to be used in domains of interest for the SAT and pseudo-Boolean communities. We show that the use of these techniques have viable processing time. Experimental results were run, analyzed and we have shown that the pre-processing time keeps the total time, to pre-process and solve the formula, viable in most cases.

We applied the proposed method on the problem virtual machine consolidation. The generated formulae for this study are being used as a domain for the pseudo-Boolean competition of 2015.

CAPÍTULO 1

INTRODUÇÃO

Esta tese explora técnicas de pré-processamento de fórmulas pseudo-Booleanas, herdadas de avanços gerados na comunidade de Pesquisa Operacional no pré-processamento de fórmulas de programação linear inteira e programação linear inteira mista.

No contexto deste trabalho um procedimento de decisão é um algoritmo que verifica a existência de alguma interpretação para uma fórmula lógica que a torne verdadeira, ou seja, se uma fórmula lógica é satisfatível ou insatisfatível. Satisfatibilidade Booleana (SAT) é o problema de decidir se uma fórmula é satisfatível, isto é, verificar se há uma valoração para as variáveis da fórmula que a torne verdadeira. O problema SAT é de grande importância em várias áreas da Ciência da Computação, incluindo teoria da computação, inteligência artificial e verificação de hardware e software. SAT foi o primeiro problema provado NP-Completo [13] e nenhum algoritmo eficiente é conhecido para resolvê-lo. No entanto muitos resultados relevantes foram obtidos na última década [49, 52, 26].

O formalismo pseudo-Booleano, embora não seja estritamente Booleano, ainda fica muito próximo às funções Booleanas. Esta representação em um sentido mais lato é uma função que mapeia n valores Booleanos em um número real. Em nosso escopo será tratado apenas o mapeamento para números inteiros.

As funções pseudo-Booleanas são estudadas desde os anos de 1960, [31] mas no contexto de Pesquisa Operacional e Programação 0-1. E dada a proximidade destas áreas, é possível que os resolvedores consigam tirar proveito dos avanços da comunidade de Satisfatibilidade e da comunidade de Pesquisa Operacional.

As formulações de problemas, utilizando representação para Satisfatibilidade, tem crescido ao longo das últimas décadas devido ao grande avanço no tempo de decisão dos resolvedores. Muitas técnicas foram refinadas e os resolvedores conseguem decidir instâncias bem maiores que anteriormente aos avanços, tal como o retrocesso não-cronológico [48] e

propagação rápida [52]. Os métodos de busca podem, agora, explorar o espaço de busca de forma sistemática ou se mover por ele de forma não sistemática. Com esses avanços, os resolvedores fornecem um ferramental muito importante para solução de problemas oriundos de instâncias do mundo real, como é o caso de verificação automática de software e hardware, e problemas de planejamento.

Os resolvedores SAT desenvolvidos atualmente utilizam duas abordagens principais: procedimentos de busca baseados no algoritmo clássico Davis-Putnam-Logemann-Loveland (DPLL) [17, 16] e procedimentos heurísticos de busca local [60]. Heurísticas de busca local, em geral, não resultam em algoritmos completos, ou seja, não há garantia que esses algoritmos encontrem uma valoração satisfatível caso ela exista, nem que provem que uma fórmula é insatisfatível. Os algoritmos não completos não podem ser utilizados na verificação, pois é necessária a garantia da satisfatibilidade da fórmula. Alguns resolvedores não completos são GSAT [60], WALKSAT [59] e POLSAT [62].

A maioria dos resolvedores SAT precisa que a fórmula de entrada, ou instância do problema, esteja na Forma Normal Conjuntiva (CNF do inglês *Conjuctive Normal Form*). Esse formato representado por uma conjunção de disjunções facilita o processo de implementação do resolvedor SAT. Dada uma valoração σ para um conjunto de variáveis da fórmula, o algoritmo *Boolean Constraint Propagation* (BCP) determina se σ torna a fórmula falsa ou provê um conjunto de implicações lógicas.

Apesar do uso intensivo de fórmulas em CNF, as fórmulas típicas de aplicações industriais não estão necessariamente nesse formato. As fórmulas que não estão em CNF chamamos de não-clausais. Para verificar a satisfatibilidade de uma fórmula não-clausal ϕ usando um resolvedor SAT para CNF temos que converter ϕ. A conversão é feita introduzindo novas variáveis [64], resultando em uma nova fórmula ϕ' que é equivalente a ϕ em relação a propriedade de satisfatibilidade, ou seja, equi-satisfatível. Outra maneira de converter uma fórmula qualquer em CNF é aplicando equivalências lógicas como: as leis de De-Morgan, lei da dupla negação e a propriedade distributiva. Neste caso a fórmula equivalente obtida pode possuir um número exponencial de cláusulas em relação ao número de variáveis. Enquanto que na conversão com introdução de novas variáveis, a fórmula

cresce apenas linearmente em relação ao número de variáveis, impactando diretamente na memória para representar a fórmula bem como no desempenho do BCP.

Para evitar as transformações das fórmulas não-clausais em CNF, uma família de resolvedores SAT atuam diretamente na representação original do problema. Esses resolvedores são chamados de resolvedores SAT não-clausais [54].

A representação pseudo-Booleana é bastante rica, e é possível modelar problemas reais facilmente utilizando essa abordagem. Apesar dessa representação ser ligeiramente diferente das representações tradicionais de SAT, ficando mais próxima da representação de programação linear inteira(ILP), as técnicas utilizadas para resolver os problemas são bastante intercambiáveis, possibilitando o aproveitamento dos avanços de SAT nas últimas décadas.

Neste trabalho, discutimos aplicações de técnicas de pré-processamento de fórmulas que causam algum impacto no tempo de decisão e otimização em um conjunto definido de fórmulas. Estas fórmulas são oriundas das competições periódicas para avaliar os avanços dos resolvedores pseudo-Booleanos, de satisfatibilidade e da representação SAT de problemas de planejamento. No escopo das técnicas de pré-processamento, resgatamos técnicas amplamente discutidas por Guignard e Spielberg [30] e Savelsbergh [58] e que posteriormente tiveram parte de suas discussões resgatadas por Dixon [20, 21, 24, 23, 22].

Neste contexto trabalhamos nesta tese de acordo com a seguinte hipótese:

Hipótese. *É possível aplicar algum método de pré-processamento em fórmulas pseudo-Booleanas que cause um impacto positivo no tempo de resolução dos resolvedores pseudo-Booleanos modernos. Ou seja, que cause uma redução no tempo de processamento do resolvedor para a fórmula pré-processada.*

1.1 Contribuição

Para verificar tal hipótese foi desenvolvido o pré-processador *pBCR*, sendo um acrônimo para *pseudo-Boolean Constraint Reduction.* Nele dividimos o método de pré-processamento em quatro etapas e são elas:

1. *Sondagem* - Etapa de sondagem dos literais da fórmula;

2. *Aprendizado* - Etapa de aprendizado, onde novas restrições são injetadas na fórmula;

3. *Fortalecimento* - Etapa de fortalecimento das restrições;

4. *Subjugação* - Etapa de subjugação de restrições.

Além do *pBCR*, este trabalho também contribuiu com uma formulação em pseudo-Boolean para consolidação de máquinas virtuais, problema real e bastante estudado na comunidade de sistemas distribuídos, que tem por objetivo alocar um conjunto de máquinas virtuais, executando em um conjunto de hardware disponível, cada um (virtuais e hardware) com sua disponibilidade e requisitos de recursos de processamento e memória RAM. Além disso, o problema ainda deve ser otimizado de forma a reduzir a quantidade de hardware utilizado. Este trabalho teve, como resultado, dois artigos publicados: "On Modeling Virtual Machine Consolidation to pseudo-Boolean Constraints" [56] publicado no IBERAMIA 2012 e "PBFVMC: A new pseudo-Boolean Formulation to Virtual Machine Consolidation" [55] publicado no BRACIS 2013.

1.2 Organização do Texto

No capítulo 2 são apresentadas as técnicas para resolvedores de SAT, pseudo-Boolean e programação inteira mista.

O capítulo 3 trata da revisão dos métodos de pré-processamento em fórmulas SAT, pseudo-Boolean e programação inteira mista.

O desenvolvimento e os passos do método *pseudo-Boolean Constraint Reduction* (pBCR) é tratado no capítulo 4, detalhando todas as etapas de pré-processamento, bem como a comparação com as técnicas que foram utilizadas da Pesquisa Operacional e adaptadas para o uso em restrições pseudo-Booleanas.

No capítulo 5 é apresentada a avaliação experimental do método, utilizando diversas fórmulas oriundas de planejamento e competições de SAT e pseudo-Boolean.

O capítulo 6 apresenta a modelagem pseudo-Booleana para a consolidação de máquinas virtuais, sendo um estudo de caso da aplicação de pseudo-Boolean à problemas reais.

No capítulo 7 a conclusão do trabalho é apresentada, bem como os trabalhos futuros.

CAPÍTULO 2

TÉCNICAS PARA RESOLVEDORES SAT, PSEUDO-BOOLEAN E PROGRAMAÇÃO INTEIRA MISTA

Este capítulo trata das técnicas atuais para a resolução de instâncias SAT, pseudo-Booleanas e Programação Inteira, além de uma breve introdução à lógica e notações que serão utilizadas ao longo de toda a tese.

2.1 Introdução à Lógica Proposicional

Esta seção apresenta definições básicas da lógica proposicional para o uso em satisfatibilidade. Serão tratadas as notações, as formas de representação e a conversão entre as formas de representação.

Uma lógica proposicional é um sistema formal no qual fórmulas representam sentenças *declarativas*, ou proposições, que podem ser verdadeiras ou falsas, mas nunca ambas. "O céu é azul." e "A resposta é 42." são exemplos de proposições.

Partindo das proposições, é possível criar sentenças compostas utilizando os conectivos lógicos. Cinco são os conectivos lógicos: $\neg$ (negação), $\wedge$ (e), $\vee$ (ou), $\Rightarrow$ (se ... então) e $\Leftrightarrow$ (se e somente se). A junção de duas proposições por um conectivo forma uma sentença composta. De maneira geral, sentenças são compostas por outras sentenças e operadores. Dessa maneira se torna possível a representação de ideias mais complexas. Para a lógica proposicional, proposições, simples ou compostas, são definidas como fórmulas bem formadas ou, simplesmente, *fórmulas*. As regras que definem recursivamente uma fórmula são as seguintes:

- Um átomo é uma fórmula.
- Se α é uma fórmula, então $(\neg\alpha)$ também é uma fórmula.
- Sendo α e β fórmulas, então $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \Rightarrow \beta)$ e $(\alpha \Leftrightarrow \beta)$ também serão.

- Todas as fórmulas são geradas pela aplicação das regras anteriores.

Variáveis proposicionais são denotadas como $x_1, ..., x_n$ ou por letras do alfabeto em minusculas como p, q, r e podem receber valores verdade *verdadeiro* (também, V ou 1) ou *falso* (também, F ou 0). O valor verdade assinalado em uma variável x é denotado como $v(x)$. Um *literal* l é uma variável x ou a sua negação $\neg x$. Uma *cláusula* c é a disjunção de literais. Uma *fórmula* CNF θ é a conjunção de cláusulas. Uma cláusula é dita *satisfeita* quando pelo menos um de deus literais assumiu o valor V e *não-satisfeita* quando todos os seus literais assumiram o valor F. *Cláusula Unitária* é uma cláusula representada por apenas um literal. *Literal Puro* é o literal que aprece em apenas uma forma em toda a fórmula, ou seja, a variável x aparece apenas na forma x ou $\neg x$ na fórmula.

O problema de satisfatibilidade consiste em decidir se existe uma valoração que torna a fórmula verdadeira, ou seja, uma atribuição de valores verdade para as variáveis da fórmula que tornem a fórmula satisfeita.

2.2 Satisfatibilidade

O algoritmo Davis-Putnam-Logemann-Loveland (DPLL) [16] foi desenvolvido por Martin Davis, George Logemann e Donald W. Loveland sendo um refinamento do algoritmo desenvolvido por Martin Davis e Hilary Putnam em 1960 [17]. O algoritmo DPLL é usado como base dos principais resolvedores da atualidade, alguns deles são: GRASP [49], ZCHAFF [52] e MINISAT [26].

O DPLL é um algoritmo simples que escolhe uma variável da fórmula e então define tentativamente um valor verdade para ela. A fórmula é simplificada e um processo recursivo verifica se a fórmula simplificada é satisfatível. Se a fórmula simplificada não for satisfatível a simplificação é desfeita e o valor verdade da variável é trocado e o processo se repete.

Durante a década de 90 o interesse pelo problema aumentou quando alguns grupos de pesquisa apresentaram estratégias que otimizaram o processo de verificação da fórmula em uma dada valoração, além de estimar qual seria a melhor escolha de variável para

assinalar um valor verdade.

Os saltos de desempenho foram marcados por passos importantes na otimização de pedaços do algoritmo DPLL. O primeiro resolvedor a contribuir com o alto desempenho foi o SATO [71] que conseguiu otimizar o processo de avaliar as consequências de uma valoração na fórmula. Essa otimização foi atingida pelo modo em que as cláusulas foram representadas. A representação das cláusulas é feita com um vetor, onde dois marcadores são colocados: um no início do vetor, e o outro no fim do vetor. Sempre que o literal marcado pelo marcador for valorado como falso, o marcador se desloca em uma posição, de forma que, quando os marcadores se encontrem, o literal marcado por ambos será ainda não valorado e todos os outros já estão valorados como falso, é feita a inferência do literal para verdadeiro. Essa representação de marcadores resulta em um aumento de desempenho porque não é mais necessário avaliar todas as cláusulas a cada valoração.

Depois do SATO uma grande contribuição veio com o GRASP [49] (*Generic seaRch Algorithm for the Satisfiability Problem*), que introduziu o conceito de retrocesso não cronólogico sobre o algoritmo DPLL, e será tratado com detalhes na seção 2.2.5. Esse conceito do GRASP fez com que, a revisão de valoração deixasse de ser feita pela inversão da última valoração feita, e passou a ser analisada de forma mais inteligente. A análise do erro acontece com a identificação da decisão que gerou a inferência da valoração, a qual causou a inconsistência na fórmula. Vale notar que quando identificada essa causa o DPLL desfaz todas as valorações até o ponto determinado como o causador do erro.

A proposta do GRASP realmente era boa, mas com sua implementação ainda clássica do BCP, fez com que os resolvedores da época, como o SATO, ainda tivessem resultados melhores. Foi então que uma nova implementação feita em 2001, denominada ZCHAFF, conseguiu unir o que existia de melhor nas pesquisas em BCP: escolha de variável e retrocesso não-cronológico. A maior contribuição do ZCHAFF foi a otimização feita no BCP, onde as cláusulas passam a ser verificadas pelo BCP, apenas quando entram em um momento crítico de valoração, ou seja, quando apenas um literal da cláusula ainda não foi valorado e todos os outros possuem valor verdade falso. Se a cláusula não está nesse estado então o BCP, grosso modo, não irá gastar processamento verificando se a cláusula

ficou vazia.

Nas próximas seções apresentaremos a representação na Forma Normal Conjuntiva e a evolução dos resolvedores que utilizam o algoritmo DPLL como base.

2.2.1 Forma Normal Conjuntiva

A Forma Normal Conjuntiva (*Conjuctive Normal Form* - CNF) é a classe de fórmulas da lógica proposicional, que utiliza apenas os operadores lógicos de conjunção ($\wedge$), disjunção ($\vee$) e negação ($\neg$). Sendo este último aplicável apenas sobre as variáveis proposicionais.

A sua representação é um grafo direcionado acíclico (DAG) de altura 2, onde a raiz é o operador de conjunção ($\wedge$), o primeiro nível são operadores de disjunção ($\vee$) e o segundo nível as folhas, compostas por literais. A figura 2.1 apresenta o diagrama como grafo para a fórmula $(p \vee q) \wedge (q \vee \neg p) \wedge (\neg p \vee \neg q)$.

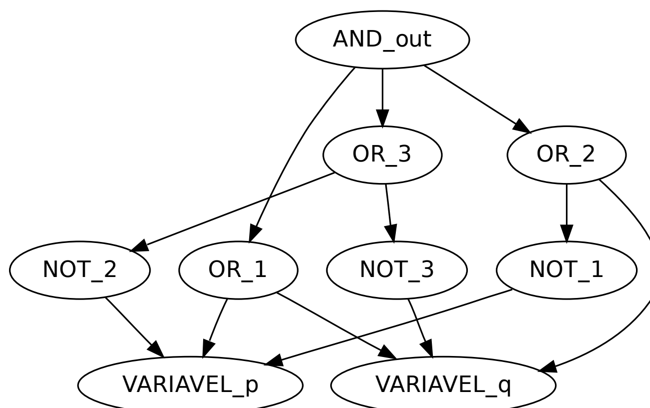

Figura 2.1: Diagrama de uma fórmula CNF.

2.2.1.1 Conversão para CNF

As fórmulas podem ser convertidas para CNF, aplicando-se as equivalências lógicas que são as leis de De-Morgan, lei da dupla negação e a propriedade distributiva. Obtém-se assim, uma fórmula equivalente que pode possuir um tamanho exponencial de cláusulas em

relação ao número de variáveis. Isto torna impraticável a sua representação computacional (devido a grande quantidade de memória requerida para a representação). Para evitar essa explosão no tamanho da fórmula, a transformação de Tseitin [64] é utilizada para a conversão.

A transformação de Tseitin consiste em adicionar novas variáveis que representam sub-fórmulas. Inserindo, na fórmula, restrições clausais que ligam o valor dessas variáveis com as sub-fórmulas que as representam, e a substituem na fórmula principal o seu representante.

Conjunções $x_1 \wedge x_2 \wedge ... \wedge x_n$ podem ser substituídas por uma única variável T_k se adicionarmos as restrições $(T_k \vee \neg x_1 \vee ... \vee \neg x_n) \wedge (\neg T_k \vee x_1) \wedge ... \wedge (\neg T_k \vee x_n)$. Essas restrições codificam a relação $T_k \Leftrightarrow (x_1 \wedge .. \wedge x_n)$.

Disjunções $x_1 \vee ... \vee x_n$ são representadas por T_l, adicionando as restrições $(\neg T_l \wedge x_1 \wedge ... \wedge x_n) \vee (T_l \wedge \neg x_1) \vee ... \vee (T_l \wedge \neg x_n)$, codificando $T_l \Leftrightarrow (x_1 \vee ... \vee x_n)$.

A transformação pode ser feita recursivamente das sub-fórmulas de nível mais baixo até a chegada na raiz da fórmula. A nova fórmula codificada cresce apenas linearmente em relação a quantidade de variáveis da fórmula original.

2.2.2 O algoritmo DPLL

Dada uma fórmula ϕ e seu conjunto de cláusulas ψ, o algoritmo DPLL consiste das seguintes regras:

Tautologia - Remove de ψ todas as cláusulas que são tautologias. O conjunto de cláusulas resultante é Insatisfatível apenas se ψ também for;

Cláusula Unitária - Se existe alguma cláusula unitária em ψ dada pelo literal ϕ, remova de ψ todas as cláusulas contendo ϕ. Se ψ for vazia, então ϕ é Satisfatível;

Literal Puro - Um literal L no conjunto de cláusulas ψ é dito puro se e somente se o literal $\neg L$ não aparece em nenhuma cláusula do conjunto ψ. Se L for puro, remova toda cláusula onde L aparece;

Ramificação - Quando nenhuma das regras acima for aplicável, ψ é reescrita na forma:

$(A_1 \vee L) \wedge \ldots \wedge (A_m \vee L) \wedge (B_1 \vee \neg L) \wedge \ldots \wedge (B_n \vee \neg L) \wedge R$

Onde A_i, B_i, R são livres de L e $\neg L$, então obtemos os conjuntos $\psi_1 = A_1 \wedge \ldots \wedge A_m \wedge R$ e $\psi_2 = B_1 \wedge \ldots \wedge B_n \wedge R$. ψ é insatisfatível se e somente se ψ_1 e ψ_2 foram insatisfatíveis.

Tendo as 4 regras descritas, percebemos que o ponto crucial está justamente na regra de Ramificação. A regra de Ramificação é o momento em que o algoritmo escolhe uma variável e define um valor. Com essa valoração o algoritmo deverá propagar o efeito resultando nos dois conjuntos de cláusulas, um definindo a variável escolhida como Verdadeira e outro conjunto com ela definida como Falsa.

A implementação computacional do DPLL não possui a regra da Ramificação estritamente como na definição, pois é fácil notar que a cada escolha de variável o conjunto de cláusulas dobra de tamanho. Não haveria memória suficiente para guardar a grande quantidade de cláusulas em fórmulas muito grandes. Por isso, as implementações são baseadas em um conceito de nível e retrocesso, ou seja, no momento da regra da ramificação é escolhida uma variável, e dado um valor (verdadeiro ou falso) e esse momento é guardado como nível de decisão i, sendo o nível de decisão de uma variável x denominado por $\delta(x)$. Se a variável recebeu o valor verdadeiro é guardado que no nível de decisão i a variável L foi decidida como verdadeira e com isso todas as cláusulas que possuem L são marcadas como removidas. Isto pode ser escrito como $x = v@d$ representando que a variável x recebeu o valor verdade v(verdadeiro) no nível de decisão d.

Quando alguma cláusula se torna Falsa a fórmula em questão também fica falsa. Mas ainda não é possível afirmar a sua insatisfatibilidade, pois o espaço de busca ainda não foi completamente esgotado. Nesse momento o algoritmo desfaz o último nível de decisão, e faz a troca de valor da variável. Se a variável L foi decidida como verdadeira, ela é marcada como Falsa, e o algoritmo continua propagando o valor. Se os dois valores de uma variável já foram testados, o nível anterior sofre a mudança de valor e assim sucessivamente.

A organização genérica do algoritmo de busca DPLL está representada no algoritmo 1. Esse algoritmo captura a essência dos resolvedores mais competitivos. O algoritmo conduz

```
    Entrada: Fórmula em CNF
    Saída: Informação SAT ou INSTATISFATIVEL
 1  enquanto verdade faça
 2    se Decide() == OK então
 3      BCP()
 4      se BCPRetornouConflito() == SIM então
 5        se Diagnostico() == NaoPodeSerResolvido então
 6          retorna INSTATISFATIVEL
 7        senão
 8          RETROCESSO()
 9    senão
10      retorna SAT
```

Algoritmo 1: DPLL

uma busca pelo espaço das possíveis valorações para as variáveis do problema. Em cada passo da busca, um valor verdade é escolhido com a função *Decide*(). E observamos que o nível de decisão d é associado com a variável valorada, representando em qual nível a variável foi valorada.

As implicações de valorações são identificadas pela função BCP() (Boolean Constraint Propagation), que é o momento em que o algoritmo procura alguma cláusula que ficou falsa, ou retorna as implicações causadas pela última decisão. Quando alguma cláusula se torna falsa o BCP() retorna o conflito que é analisado pela função *Diagnostico*(). A função de diagnóstico analisa o conflito, e identifica em qual nível de decisão foi cometido um erro, e então um retrocesso é feito, desfazendo todas valorações do nível de decisão atual, até o nível marcado como fonte do erro. A maneira na qual um conflito é analisado será explicada mais adiante no texto. A seguir veremos como as funções *Decide*() e BCP() foram trabalhadas ao longo dos anos até o modelo utilizado hoje nos principais resolvedores.

A função *Decide*() escolhe uma variável, ainda não valorada, e atribui um valor verdade, enquanto que a função BCP() percorre a fórmula aplicando a decisão feita em *Decide*(). Na descrição das 4 regras do DPLL clássico, podemos dizer que o BCP() é a aplicação das regras 1-3 enquanto que a *Decide*() é a regra da ramificação.

O BCP propaga o efeito da valoração feita até o momento, marcando os literais valorados como falso fora da cláusula e marca as cláusulas com pelo menos um literal verdadeiro como verdadeiro, excluindo essas cláusulas do conjunto de cláusulas. Sempre que o BCP é chamado, essa verificação é feita em torno da valoração parcial, sendo um dos procedimentos mais importantes e mais custosos do resolvedor. O seu consumo varia entre $80 - 90\%$ do tempo gasto pelo resolvedor [52]. Vários estudos foram feitos para melhorar o desempenho do BCP, além de estudos para a escolha da variável mais adequada para aplicar a regra da ramificação.

Cada função possui a sua equivalência nas regras do DPLL, exceto pela função *Diagnostico()* que nos resolvedores modernos tem o objetivo de analisar o estado da valoração, e identificar o motivo da fórmula estar em um estado que a torne insatisfatível. O maior avanço dessa função foi atingido com o GRASP [49] que conseguiu identificar o motivo, retroceder para a decisão que foi a base, para o estado atual de valoração e ainda adicionar ao conjunto de cláusulas, uma nova, que representa o conjunto de literais que devem sempre ficar verdadeiro. O processo de analisar, diagnosticar e adicionar uma nova cláusula será apresentado na seção 2.2.5.

A ordem de escolha de variáveis, para aplicar a regra da ramificação, pode melhorar significativamente o tempo de execução do resolvedor SAT. Se uma fórmula for satisfatível e, a cada passo do algoritmo, a variável escolhida para a ramificação, tiver seu valor verdade marcado, tal como deve ser na linha da tabela verdade, para que torne a fórmula verdadeira, o número de decisões e retrocessos feitos durante a execução do resolvedor pode ser muito menor que escolher uma variável qualquer.

Vários estudos foram feitos no sentido de se encontrar a melhor heurística de escolha de variável, mas essa escolha depende muito de como a fórmula foi gerada e mais ainda da instância do problema. Por isso é muito difícil de afirmar, o que faz com que uma heurística seja um boa estratégia em relação a outras. Em 1999, João Marques-Silva publicou um estudo com algumas heurísticas e seus resultados empíricos com diversas fórmulas e resolvedores SAT. A seguir veremos algumas heurísticas.

2.2.3 Heurísticas para ramificação

A estratégia mais simples de escolha de variável está na escolha aleatória de uma variável ainda não valorada. Essa estratégia é denominada RAND. Testes empíricos mostram que a utilização da estratégia RAND pode ser tão eficiente quanto a mais sofisticada técnica de escolha.

As heurísticas, que conseguiram ter resultado melhor que o RAND na maioria dos testes, levam em consideração informação dinâmica fornecida pelo BCP. O BCP repassa para a máquina de decisão essas informações. Dentre as heurísticas temos a BOHM [9, 47], MOM [28, 70, 47] e VSIDS [52].

A heurística BOHM (o nome vem do autor) era bastante competitiva para instâncias aleatórias em 1992 como foi descrito em [9] e a ideia é a de escolher a variável com maior vetor $(H_1(x), H_2(x), ..., H_n(x))$ em ordem lexicográfica onde $H_i(x)$ é computado da seguinte maneira:

$$H_i(x) = \alpha \times max(h_i(x), h_i(\neg x)) + \beta min(h_i(x), h_i(\neg x))$$

onde $h_i(x)$ representa o número de cláusulas não resolvidas com i literais que possuam o literal x. Com isso cada literal selecionado dá preferência em satisfazer cláusulas pequenas (quando valoradas como verdadeiras) ou reduzir o tamanho das cláusulas pequenas (quando valoradas como falsas). Os valores de α e β são dependentes da implementação da heurística, no caso de [9] os valores sugeridos são $\alpha = 1$ e $\beta = 2$.

Maximum Occurrences on clauses of Minimum size (MOM) é a heurística que, pelo nome, busca pelo maior número de ocorrências de um literal nas menores cláusulas presentes na fórmula. Para isso temos $f^*(l)$ o número de ocorrências de um literal l na menor cláusula ainda não satisfeita. Uma boa escolha de variável a ser selecionada é uma que maximiza a função:

$$(f^*(x) + f^*(\neg x)) \times 2^k + f^*(x) \times f^*(\neg x)$$

Intuitivamente a preferência é dada para variáveis x que tenham a maior quantidade

de ocorrência dos literais x ou $\neg x$ (considerando que k possua um valor suficientemente alto), e também para variáveis com várias ocorrências de seus literais x e $\neg x$.

Jeroslow e Wang em [37] propuseram duas heurísticas, chamadas *one-sided Jeroslow-Wang* (JW-OS) e *two-sided Jeroslow-Wang* (JW-TS). Para um dado literal l compute:

$$J(l) = \sum_{l \in w \wedge w \in \phi} 2^{-|w|}$$

A JW-OS valora o literal l que possui o maior $J(l)$, enquanto que a JW-TS identifica a variável x com a maior soma $J(x) + J(\neg x)$ e valora x como verdadeira se $J(x) >= J(\neg x)$, e falso caso contrário.

Mesmo tendo um grande número de heurísticas para tentar escolher a melhor variável, é importante saber como avaliar e escolher a melhor. Alguns podem considerar a melhor escolha, aquela que influencia no número de decisões que o algoritmo de busca irá fazer, pois menos decisões indicam que as tomadas foram mais inteligentes. O problema é que, cada decisão influencia o trabalho do BCP de forma diferente, ou seja, uma sequência de decisões menor pode causar mais operações no BCP do que uma sequência maior. E ainda, o custo computacional de cada heurística de escolha de variável é diferente. A melhor decisão a ser feita pode custar muito computacionalmente para ser calculada, ao passo que uma heurística simples como a RAND não consome quase processamento, e pode fazer algumas escolhas boas, e no tempo final, o resolvedor SAT pode ser menor. O fato é que, não existe uma resposta clara na literatura que indique uma estratégia que seja boa em qualquer caso.

O grupo, que desenvolveu o resolvedor ZCHAFF, apresentou uma nova estratégia denominada VSIDS (*Variable State Independent Decaying Sum*). Ela conseguiu atuar de uma forma melhor que várias outras estratégias pensadas anteriormente. Funciona da seguinte maneira:

1. todo literal tem um contador iniciado como 0;

2. quando uma cláusula nova é adicionada, o contador associado com cada literal presente na cláusula é incrementado;

3. a variável (não valorada) e seu literal com o contador mais alto é escolhida para a decisão;

4. periodicamente todos os contadores são divididos por uma constante.

A implementação dessa heurística conta com uma lista das variáveis não valoradas, ordenadas pelo valor do contador que é atualizado durante o BCP e na análise de conflitos. Dessa forma a escolha da variável é feita de modo muito rápido no momento da decisão.

O MINISAT utiliza uma abordagem semelhante, porém não distingue os literais de uma variável. A contagem é feita apenas pela variável (que é a soma dos literais). Após gravar o conflito, o contador de todas as variáveis é multiplicado por um valor menor que 1, diminuindo a atividade das variáveis com o tempo.

O ponto chave dessa estratégia é a pequena exigência de processamento, pois é atualizada apenas quando uma nova cláusula é adicionada, decorrente de um conflito. Os autores do ZCHAFF afirmam que essa estratégia melhorou o resolvedor em uma ordem de magnitude com problemas difíceis, e não prejudicou em problemas mais simples, o que pode ser visto como uma métrica para afirmar o seu sucesso.

2.2.4 BCP rápido

Motivados pela grande parcela de processamento do BCP, os autores do CHAFF decidiram otimizá-lo e, perceberam que uma cláusula entra em estado crítico de valoração, quando todos os literais foram valorados como falso e resta apenas um literal não valorado. Quando a cláusula entra nesse estado crítico o literal não valorado pode ser inferido facilmente, ou seja, deve ser valorado como verdadeiro para que a fórmula não fique com uma valoração insatisfatível.

Para determinar se uma cláusula está no estado crítico, o ZCHAFF então implementa a seguinte idéia: cada cláusula possuirá uma lista de observações de dois literais, e quando uma variável for valorada, apenas as cláusulas que possuem o literal da variável na lista de observação serão verificados. Se o literal for falso, a lista de observação mudará, fazendo com que o literal valorado como falso deixe de ser observado, sendo trocado por um outro

literal qualquer que ainda não tenha sido valorado na cláusula. Quando só existe um literal a ser escolhido, o literal restante é inferido como verdadeiro.

Quando um literal é inferido, a máquina de busca não muda de nível de decisão, pois é uma inferência feita pela decisão tomada no nível, propagando o efeito até que não existam mais inferências.

2.2.5 Aprendizado de cláusulas e retrocesso não-cronológico

A contribuição mais relevante para a resolução de satisfatibilidade foi com a introdução do algoritmo GRASP [49], onde o retrocesso deixou de ser apenas com a troca do valor verdade da última decisão e passou a ser inteligente, analisando o porquê da cláusula se tornar falsa, e voltar todos os níveis de decisão até a causa do erro. Esse processo é chamado de retrocesso não-cronológico pois o resolvedor deixa de voltar apenas um nível de decisão, e passa a voltar vários. O GRASP ainda permite que o sistema aprenda o motivo do erro, evitando, assim, a recorrência. Desse modo dizemos que o GRASP contribuiu com a análise de conflito e com o aprendizado de cláusulas.

Todos os resolvedores da atualidade fazem uso das técnicas que o GRASP introduziu, apenas variando o modo de guardar as cláusulas aprendidas ou outras pequenas modificações. Veremos agora como o GRASP permite esse retrocesso não-cronológico, e como é feito o aprendizado.

A principal modificação do GRASP no algoritmo clássico do DPLL, mostrado no Algoritmo 1, está na função $Diagnostico()$. Quando o BCP retorna conflito é feita uma análise da causa. Essa análise é feita com a criação de um grafo dirigido de implicação I, definido da seguinte forma:

1. cada vértice de I corresponde à valoração de uma variável $x : v(x)$;

2. os predecessores de um vértice $x : v(x)$ em I são valorações antecedentes $A^w(x)$ correspondente a cláusula unitária w que causou a implicação de x. As arestas dirigidas partindo do vértice $A^w(x)$ para o vértice $x : v(x)$ são nomeados com w. Vértices sem antecedentes correspondem à decisões;

3. vértices especiais de conflitos são adicionados em I para indicar a ocorrência de conflito. Os predecessores de um vértice de conflito K correspondem a valoração de variáveis que forçaram a cláusula w a ficar Falsa e são vistos como a valoração antecedente de $A^w(x)$. As arestas partindo dos vértices $A^w(x)$ para K são nomeados como w.

O nível de decisão de uma variável implicada x é relacionado com os seus antecedentes de acordo com:

$$\delta(x) = max((\delta y|(y, v(y)) \in A^w(x))$$

Com o grafo sendo atualizado a cada decisão ou implicação, temos que identificar uma valoração de conflito e efetuar alguma ação. Quando o BCP retorna um conflito, a sequência de implicações aparece convergindo para um vértice K e esse caminho é analisado para determinar o responsável pelo conflito. A conjunção das valorações conflitantes é um implicante suficiente para o conflito aparecer. A negação desse implicante gera uma função booleana f (cuja satisfatibilidade é procurada) que não existe no conjunto de cláusulas. Esse implicante é denominado cláusula de conflito induzida e provê o mecanismo para a implementação de um retrocesso não cronológico.

Quando o conflito aparece na fórmula é verificado se a decisão feita no nível de decisão corrente está presente. Se estiver, o seu valor é trocado, e nesse momento a variável deixa de ser uma decisão, e a implicação dela está nas outras decisões presentes na cláusula de conflito induzido. E o processo continua.

Quando, as variáveis presentes na cláusula de conflito induzido são todas de um nível mais baixo que o nível corrente, então o algoritmo escolhe a que foi decidida no nível mais alto, dentro da cláusula de conflito induzido. Quando o maior nível é $d - 1$ (sendo d o nível corrente) o algoritmo faz um retrocesso cronológico, quando for menor que $d - 1$ o algoritmo então está fazendo um retrocesso não-cronológico.

Para evitar que o mesmo erro seja feito várias vezes durante o processo de busca, sempre que o retrocesso for feito, a cláusula de conflito induzido é adicionada ao conjunto

de clásulas. Esse processo é denominado aprendizado de cláusula.

Exemplificando o processo, tomemos a fórmula:

$$(\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_5 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_6 \vee x_5)$$

E tomemos a seguinte valoração conflitante:

x_6 verdadeira no nível 1 ($v@1$);

x_5 verdadeira no nível 1 ($v@1$);

x_4 falsa no nível 2 ($f@2$);

x_1 verdadeira no nível 3 ($v@3$);

x_2 verdadeira no nível 3 ($v@3$);

x_3 verdadeira e falsa no nível 3 (*conflito*);

O resolvedor fará uma busca no grafo, a partir dos nós, que indicam x_3 e retrocederão até que seja encontrada um nó que não foi implicação do nível de decisão corrente. Na figura 2.2, podemos identificar um corte no grafo, que representa o local onde podemos extrair os componentes da cláusula de conflito induzido que envolvem as variáveis x_2 e x_5. Portanto a cláusula aprendida é negação das valorações das variáveis aprendidas, ficando: $(\neg x_2 \vee \neg x_5)$. Outra cláusula aprendida poderia ser $(x_4 \vee \neg x_1 \vee \neg x_6)$, porém essa última é menos sucinta que a primeira pois x_2 dá origem a x_4 e x_1.

2.2.6 Outras otimizações

Além das técnicas apresentadas, existem ainda mais dois procedimentos que podem ser usados, para evitar que o resolvedor fique em uma busca sem progresso substancial, e ainda um limpador de cláusulas aprendidas, que não possuem utilidade, e apenas acrescentam uma perda de desempenho para o BCP e para o consumo de memória.

Os resolvedores, que implementam o aprendizado de conflitos, passaram a efetuar reinícios automáticos do processo de busca: periodicamente após uma quantidade definida

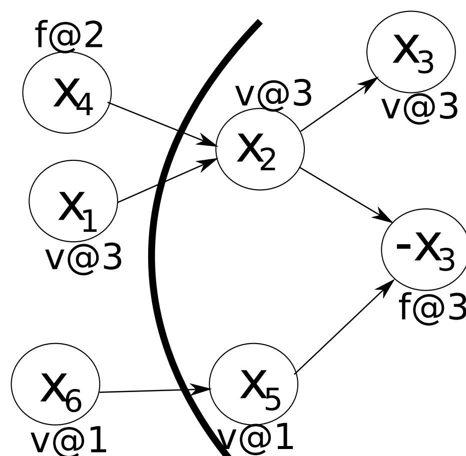

Figura 2.2: Grafo de decisão com conflito na variável x_3

de decisões, ou quantidade de cláusulas aprendidas. O algoritmo reinicia toda a busca guardando apenas o conjunto de cláusulas aprendidas.

O reinício é feito para evitar que a busca fique presa em um sub-espaço, onde nenhum progresso substancial possa ser feito. Experimentos com várias heurísticas para se determinar o momento de fazer o reinício podem ser vistos em [34].

Muitos resolvedores colocam um limite em relação a quantidade de cláusulas que são mantidas, para evitar uma explosão no uso de memória. As técnicas variam entre as implementações; alguns resolvedores removem cláusulas apenas durante os reinícios, outros mesclam algumas cláusulas ou, tentam identificar quais cláusulas são subconjuntos de outras, e guardam apenas uma delas.

O ZCHAFF, ao adicionar uma cláusula, faz uma análise para determinar em que ponto no futuro ela pode ser removida. A métrica usada é a relevância, ou seja, quando mais que N (onde N varia entre 100 e 200) literais na cláusula ficam sem valoração pela primeira vez, a cláusula será marcada para remoção.

2.3 Programação Inteira Mista

Definição 1. *Dada uma matriz $A \in \mathbb{R}^{m \times n}$, vetores $b \in \mathbb{R}^m$, e $c \in \mathbb{R}^n$, e o subconjunto $I \subseteq N = \{1, ..., n\}$, o programa inteiro misto $MIP = (A, b, c, I)$ deve resolver*

$$c^* = min\{c^T x | Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \text{ para todo } j \in I\}$$

Os vetores no conjunto $X_{MIP} = \{x \in \mathbb{R}^n | Ax \leq b, x_j \in \mathbb{Z} \text{ para todo } j \in I\}$ são chamados de soluções factíveis *do MIP. Uma solução factível $x^* \in X_{MIP}$ de um MIP é chamada de ótima se o seu objetivo satisfaz $c^T x^* = c^*$.*

Os resolvedores MIP geralmente tratam restrições simples limitadas $l_j \leq x_j \leq u_j$ com $l_j, u_j \in \mathbb{R} \cup \{\pm\infty\}$ separadamente das restrições restantes. Em particular, variáveis inteiras com limites $0 \leq x_j \leq 1$ tem um papel especial nos algoritmos de solução e são bastate importantes para modelar decisões do tipo "sim/não". Nos referimos ao conjunto de *variáveis binárias* com $B := \{j \in I | l_j = 0 \text{ e } u_j = 1\} \subseteq I \subseteq N$. Além disso, denotamos as variáveis contínuas por $C := N\ I$.

Os casos especiais comuns de MIP são programas lineares (LPs) com $I = \emptyset$, programas inteiros (IPs) com $I = N$, programas binários mistos (MBPs) com $B = I$, e programas binários (BPs) com $B = I = N$. O problema de satisfatibilidade é um caso especial de BP sem função objetiva. Sendo SAT em NP-Completo, BP, IP, e MIP são NP-Hard.

Os métodos apresentados a seguir são muito conhecidos e utilizados em programação inteira, e nas técnicas clássicas de pseudo-Boolean, e podem ser vistos como uma forma de automação de sistemas de planos de corte. No entanto, o tipo de automação difere-se dos métodos de resolução em SAT e pseudo-Boolean, pelos últimos serem inspirados em idéias e técnicas da comunidade de inteligência artificial. Os algoritmos clássicos de banch-and-bound e branch-and-cut serão rapidamente discutidos nas seções a seguir.

2.3.1 Branch-and-Bound

A abordagem clássica de resolver problemas de programação inteira é com branch-and-bound. É um algoritmo de backtracking sistemático no estilo de árvore, semelhante aos

algoritmos discutidos na seção anterior. Diferentemente de problemas de satisfatibilidade, os problemas de programação inteira são de otimização. A principal técnica de poda usada pelo branch-and-bound é feita para eliminar soluções factíveis, mas sub-ótimas. A idéia por trás do branch-and-bound é encontrar uma solução inteira factível o quanto antes, e utilizar esta solução para podar as áreas improdutivas do espaço de busca.

Os algoritmos de branch-and-bound, assim como os algoritmos no estilo do DPLL, usam ramo de decisões para particionar iterativamente o espaço de busca na árvore de busca. Um nó n na árvore corresponde a uma série de ramos de decisões, uma para cada nó no caminho da raiz até n. Diferentemente das decisões nas árvores no estilo DPLL, os ramos de decisão numa árvore branch-and-bound pode não corresponder ao valor da variável.

Em cada nó da árvore de busca, uma relaxação do problema é resolvida. Uma relaxação é uma versão mais fácil do problema que pode ser resolvida em tempo polinomial. O conjunto da solução relaxada sempre é um superconjunto da solução do problema original. Por esta razão sempre provê um limite superior do problema original (assumindo um problema de maximização). Uma boa relaxação é uma boa aproximação da solução verdadeira.

Na árvore do branch-and-bound, a solução para relaxação pode prover dois tipos de limites. Se em algum momento durante a busca a relaxação identifica uma solução factível, esta solução pode ser armazenada como uma solução incumbente. A solução incumbente armazena a melhor solução factível encontrada até o momento e é atualizada a cada nova solução mais próxima do ótimo. Adicionalmente, qualquer solução para o problema relaxado em um nó dado, provê o limite superior para todos os subproblemas gerados abaixo daquele nó. Se a solução para a relaxação em algum nó for menor que a solução incumbente, então a subárvore abaixo pode ser removida.

A maneira mais comum de relaxação usada para IP é a relaxação linear que consiste em remover ou "relaxar" as restrições inteiras do problema e então resolver o problema

de programação linear associado. Um problema de LP pode ser estabelecido como:

$$maximize\ c^T x$$
$$sujeito\ a:\ \ Ax \leq b$$

onde, $A \in \mathbb{R}^{m\times n}$ é uma matriz de reais $m \times n$, e $b \in \mathbb{R}^m$ e $c \in \mathbb{R}^n$ são vetores reais. O problema de programação linear é resolvido em tempo polinomial, pelos métodos de elipsóide e pontos interiores. O método mais comum é o simplex, que não é polinomial mas é muito rápido na prática. A solução para a relaxação linear provê os limites para poda na árvore de busca.

Cada desigualdade linear $ax \leq b$ em um problema de programação linear define um semi-espaço no espaço $n - dimensional$. A intersecção dos semi-espaços é um poliedro, ou um politopo se for limitado. Soluções para os problemas relaxados são pontos no politopo, enquanto que soluções inteiras são um subconjunto de um politopo. A solução ótima para o problema relaxado sempre será um ponto extremo do politopo. Existem muitas referências que discutem as técnicas de programação linear [1, 68, 12].

2.3.2 Branch-and-Cut

Branch-and-Cut é uma variação do branch-and-bound com a habilidade adicional de gerar planos de corte a cada nó. Dado um problema de programação linear, um corte ou plano de corte é uma desigualdade linear, que é satisfeita por soluções integrais do problema. No contexto de branch-and-cut, estamos interessados em gerar um tipo específico de corte, chamado corte de separação. Um corte de separação é um corte, que é satisfeito em todas soluções integrais, mas é insatisfatível pela solução linear relaxada.

Existem diversos trabalhos que mostram análises que são computacionalmente viáveis quando aplicadas. É possível encontrar detalhes das técnicas em [1, 40].

2.4 Satisfatibilidade Pseudo-Booleana

Em uma definição mais direta, uma função pseudo-Boolean é aquela que mapeia valores Booleanos para um número real. O termo pseudo-Boolean é dado porque essas funções não são Booleanas, mas ainda permanecem muito próximas de funções estritamente Booleanas [48, 44, 57]. Em uma fórmula pseudo-Boolean, variáveis possuem domínio Booleano e as restrições, conhecidas como restrições pseudo-Boolean (PB) [57], são desigualdades linerares com coeficientes inteiros. Em otimização pseudo-Boolean uma função de custo é adicionada à fórmula.

As restrições pseudo-Boolean oferecem uma maneira mais expressiva e natural de representação que cláusulas e, ainda, é um formalismo que fica muito próximo ao problema da Satisfatibilidade (SAT) [48, 44] e se beneficia muito com avanços recentes dos resolvedores SAT.

Simultaneamente, resolvedores PB se beneficiam da vasta e forte experiência da programação linear inteira (*Integer Linear Programming, ILP*) e mais especificamente da programação 0-1. Isso é particularmente verdadeiro, quando problemas de otimização são considerados. Regras de inferência permitem resolver problemas polinomialmente quando representados com restrições PB, enquanto que, resolução dos problemas representados com cláusulas requerem um número exponencial de passos. Restrições PB aparecem como um meio termo entre o poder de representação no formalismo usado e a dificuldade de resolver os problemas nesse formalismo [57].

2.4.1 Representação Pseudo-Booleana

Em uma fórmula pseudo-booleana as variáveis estão no domínio Booleano e as restrições são desigualdades lineares com coeficientes integrais.

Uma restrição pseudo-booleana é definida como um conjunto finito de variáveis Booleanas x_j. As variáveis Booleanas podem receber apenas dois valores, *falso* e *verdadeiro* que podem ser representados por $\{\mathbb{F}, \mathbb{V}\}$ ou simplesmente $\{0, 1\}$. Um literal l_j é uma variável Booleana x_j ou sua negação $\neg x_j$. Um literal positivo valora para $\mathbb{V}$ apenas se

sua variável correspondente x_j também for verdadeira, o literal negativo valora para $\mathbb{V}$ se e somente se a sua variável correspondente for falsa.

Nas restrições pseudo-Boleanas, os literais sempre são atribuídos a coeficientes inteiros e quando o coeficiente for 1 ele, geralmente, é omitido na representação. A multiplicação de uma constante inteira por uma variável Booleana é definida como da maneira convencional, quando as variáveis são representadas como 0 ou 1. Quando as variáveis Booleanas são representadas como $\mathbb{F}$ ou $\mathbb{V}$ a multiplicação é definida equivalente por $\forall a \in \mathbb{Z}, a.\mathbb{V} = a$ e $a.\mathbb{F} = 0$.

Restrições Lineares

Uma restrição pseudo-Boolean linear tem a seguinte forma:

$$\sum_j a_j l_j \triangleright b$$

onde a_j e b são constantes inteiras, l_j são literais e $\triangleright$ é um dos operadores relacionais clássicos ($=, >, \geq, <$ ou $\leq$).

O lado direito da restrição geralmente é chamada de grau da restrição.

O operador de soma e os operadores relacionais possuem o seu significado usual da matemática. Uma restrição está satisfeita quando os termos da direita e esquerda valoram para inteiros, que satisfaçam o operador relacional.

Qualquer restrição que pode ser reescrita da forma $\sum_j a_j l_j \triangleright b$ é considerada como uma restrição pseudo-Boolean linear.

Exemplo 1. *Como um primeiro exemplo de uma restrição linear pseudo-Boolean* $3x_1 + 4\overline{x_2} + 5x_3 \geq 7$. *Sendo satisfeita quando* $x_1 = 1, x_2 = 0$ *ou* $x_1 = 1, x_3 = 1$ *ou* $x_2 = 0, x_3 = 1$.

Outros exemplos:

$$x_1 + x_2 + \overline{x_4} < 3$$

$$5 + x_1 = 6 - x_2$$

$$8x_4 + 4x_3 + 2x_2 + x_1 \leq 8y_4 + 4y_3 + 2y_2 + y_1$$

Restrições não-lineares

Uma restrição pseudo-Boolean tem a seguinte forma:

$$\sum_j a_j.(\prod_k l_{j,k}) \rhd b$$

onde a_j e b são constantes inteiras, $l_{j,k}$ são literais e $\rhd$ é um dos operadores relacionais clássicos $(=, >, \geq, <$ ou $\leq)$. A semântica do produto dos literais é definido de forma natural as variáveis Booleanas são representadas como $\{0, 1\}$ e correspondem ao E lógico quando representadas como $\{\mathbb{F}, \mathbb{V}\}$. A restrição é satisfeita quando os termos esquerdo e direito valoram os inteiros que satisfaçam o operador relacional. Existem maneiras para linearizar as restrições, evitando que os resolvedores precisem atuar diretamente na forma não linear.

Exemplo 2. *As duas restrições abaixo são não-lineares:*

$$7x_1x_2 + 3x_1 + x_3 + 2x_4 \geq 8$$

$$7\overline{x_2} + 3x_1\overline{x_3} + 2x_4x_2x_1 \geq 8$$

A restrição $x_1 + \overline{x_1}x_2 \geq 1$ *é satisfeita quando* $x_1 = 1$ *ou quando* $x_1 = 0$ *e* $x_2 = 1$*. A restrição* $x_1 + \overline{x_1}x_2 \geq 2$ *é insatisfatível.*

Restrições de Cardinalidades

Uma restrição de cardinalidade é restrição no número de literais que são verdadeiros dentre o conjunto de literais. Três restrições de cardinalidade podem ser definidas. A restrição $atleast(k, \{x_1, x_2, \ldots, x_n\})$ é verdadeira se e somente se pelo menos k literais dentre $x_1, x_2, \ldots, x_n$ forem verdadeiros. A restrição $atmost(k, \{x_1, x_2, \ldots, x_n\})$ é verdadeira se e somente se no máximo k literais forem verdadeiros. A restrição $exactly(k, \{x_1, x_2, \ldots, x_n\})$ é verdadeira se e somente se exatamente k literais forem verdadeiros. Algumas relações são

óbvias entre as restrições e são elas: $atmost(k, \{x_1, x_2, \ldots, x_n\}) \equiv atleast(n-k, \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\})$ e $exactly(k, \{x_1, x_2, \ldots, x_n\}) \equiv atmost(k, \{x_1, x_2, \ldots, x_n\}) \wedge atleast(k, \{x_1, x_2, \ldots, x_n\})$. No entanto a restrição *atleast* é suficiente para expressar qualquer restrição de cardinalidade.

A restrição de cardinalidade $atleast(k, \{x_1, x_2, \ldots, x_n\})$ em uma restrição pseudo-Boolean é escrita como $x_1 + x_2 + \ldots + x_n \geq k$ (e a restrição $atmost(k, \{x_1, x_2, \ldots, x_n\})$ fica como $x_1 + x_2 + \ldots + x_n \leq k$). Reciprocamente, uma restrição pseudo-Boolean onde todos os coeficientes são iguais como em $\sum_j ax_j \geq b$ é na verdade uma restrição de cardinalidade: $\sum_{j=1}^{n} ax_j \geq b \equiv atleast(\lceil b/a \rceil, \{x_1, x_2, \ldots, x_n\})$ ($\lceil b/a \rceil$ é o menor inteiro maior ou igual a b/a). Claramente, uma restrição de cardinalidade é um tipo especial de restrição pseudo-Boolean.

Cláusulas

A cláusula $x_1 \vee x_2 \vee \ldots \vee x_n$ é por definição equivalente a restrição de cardinalidade $atleast(1, \{x_1, x_2, \ldots, x_n\})$ que, por sua vez, é equivalente a restrição pseudo-Boolean $x_1 + x_2 + \ldots + x_n \geq 1$. Portanto, cláusulas são um subconjunto de restrições de cardinalidade que por sua vez são um subconjunto de todas as restrições possíveis pseudo-Boolean. Restrições pseudo-Boolean generalizam restrições de cardinalidade que já são uma generalização de cláusulas.

Normalização de Restrições pseudo-Boolean

Embora diversos operadores relacionais possam ser utilizados na definição das restrições pseudo-Boolean, todas as restrições pseudo-Boolean podem ser normalizadas para a seguinte forma:

$$\sum_{j=1}^{n} a_j l_j \geq b, \quad a_j, b \in \mathbb{N}_0^+ \tag{2.2}$$

onde l_j representa ou a variável x_j ou o seu complemento $\overline{x_j}$ e todos os coeficientes a_j e o lado direito b são não negativos [4] (uma função pseudo-Boolean apenas com

coeficientes positivos é chamada de posiform [10]). Além disso, restrições de igualdade ou outros tipos de desigualdades também podem ser transformadas em tempo linear em uma restrição do tipo maior ou igual.

Para normalizar uma restrição pseudo-Boolean, é possível começar modificando para uma restrição maior ou igual. Se o operador da restrição for $>$, basta adicionar 1 no lado direito e mudar o símbolo da desigualdade para $\geq$. Quando a restrição for do tipo $<$, basta subtrair 1 do lado direito e mudar para $\leq$. Note que essas modificações são válidas porque todos os coeficientes a_j são inteiros. Para equações de igualdade, pode-se substituí-la por duas restrições de tipos $\leq$ e $\geq$. Finalmente as restrições do tipo $\leq$ multiplicar por -1 ambos os lados e mudar o sinal para $\geq$.

Para o próximo passo resta modificar todos coeficientes e o lado direito para não-negativos. Para todos os literais x_j e $\overline{x_j}$ com coeficientes negativos, e substituir por $1 - \overline{x_j}$ e $1 - x_j$, respectivamente. Todos os coeficientes agora serão não-negativos e por manipulação algébrica, os termos constantes podem ser movidos para o lado direito da restrição. Se uma restrição obtida possuir o lado direito não-positivo, então a restrição é trivialmente satisfeita.

Esse procedimento permite mapear em tempo linear todas as restrições de uma fórmula pseudo-Boolean em uma formulação normalizada.

2.4.2 Propagação

A propagação, assim como em satisfatibilidade em CNF, é um ponto crítico dos resolvedores pseudo-Booleanos. A técnica utilizada pelos resolvedores em CNF não pode ser traduzida de forma exatamente igual, uma vez que nem sempre basta vigiar apenas dois literais na restrição, já que pode ser necessária uma composição de vários literais para garantir que a restrição ainda possa ser satisfeita. Nesta seção veremos duas abordagens de propagação, uma com contadores e outra com literais vigiados.

```
Entrada: Fórmula pseudo-Boolean P
Saída: SAT ou UNSAT para a fórmula P
1 enquanto verdade faça
2   se Solucao_Encontrada(P) então
3     retorna SAT
4   senão
5     Decida(p)
6   enquanto BCP(P) == CONFLITO faça
7     se Diagnostico(P)== CONFLITO então
8       retorna UNSAT
```

Algoritmo 2: Algoritmo genérico para resolver pseudo-Boolean

Método Baseado em Contadores

Definição 2. *Seja c um conjunto de literais e P uma valoração parcial. Definimos os conjuntos:*

$$S_p(c) = \{x_i | x_i \in c \ e \ x_i \in P\}$$

$$V_p(c) = \{x_i | x_i \in c \ e \ \neg x_i \notin P\}$$

onde $S_p(c)$ representa os literais em c que estão satisfeitos em P e $V_p(c)$ representa os literais em c que não são nem satisfeitos nem não valorados em P.

Definição 3. *Seja c uma restrição pseudo-Boolean na forma $\sum a_i x_i \geq k$, T um subconjunto de literais em c, e P uma valoração parcial dos valores das variáveis. As contagens current e possible são definidas como:*

$$current(T, P) = \sum_{i | x_e \in S_p(T)} a_i - k$$

$$possible(T, P) = \sum_{i | x_e \in V_p(T)} a_i - k$$

A contagem *current* soma os pesos dos literais satisfeitos em T e subtrai k. Se o valor $current(c, P)$ for maior ou igual a zero, então a restrição está satisfeita. Se o valor for negativo, a restrição ainda não está satisfeita. A contagem *possible* soma todos os pesos de todos os literais satisfeitos e não valorados, em T e subtrai k. Se o valor $possible(c, P)$ for maior ou igual a zero, então é possível satisfazer a restrição. Se o valor for negativo,

significa que não é mais possível satisfazer a restrição c. Os contadores *current* e *possible* podem ser mantidos para cada restrição, e ajustados a cada mudança em P.

Literais Vigiados

Muito é discutido sobre os literais vigiados e, de fato essa foi uma das grandes revoluções na maneira de resolver satisfatibilidade. Foi analisado e argumentado na seção 2.2.4. O método de literais vigiados mantém dois ponteiros por cláusula, com a propriedade dos dois literais não estarem valorados, ou que pelo menos um deles esteja satisfeito. Enquanto essa propriedade for mantida, a cláusula não pode ser unitária. Primeiramente essa regra será generalizada para restrições de cardinalidade, como feito em [24].

Definição 4. *Dado uma restrição de cardinalidade c na forma $\sum_i x_1 \geq k$ and dado S um subconjunto de literais em c. S é um conjunto vigiado de c em valoração parcial P se e somente se as seguintes propriedades em S são verdadeiras:*

1. *possible*$(S, P) \geq 1$
2. *current*$(S, P) \geq 0$

Proposição 1. *Se S é um conjunto vigiado de c dentro de P então c não pode ser unitário.*

Seja S um conjunto vigiado para uma restrição c em valoração parcial P. Considere o que acontece quando o literal $x_i \in S$ é valorado desfavoravelmente adicionando $\overline{x_i}$ em P formando uma nova valoração P'. Existem duas maneiras de de obter o conjunto vigiado de c dentro de P': criando um novo conjunto $S' = S - \{x_i\} \cup \{x_j\}$ tal que $x_j \in c, x_j \in V_{P'}(c)$ e $x_j \notin S$, ou mantendo o mesmo conjunto vigiado S e estender a valoração parcial P' marcando os literais restantes em S para 1. Se $k = 1$, então $|S| = 2$, e a definição 4 reduz para a definição de conjunto vigiado clausal.

Uma versão de literais vigiados para restrições pseudo-Boolean genéricas também é possível.

Definição 5. *Dada c uma restrição pseudo-Boolean na forma $\sum_i (a_i x_i) \geq k$ e seja S um subconjunto de literais em c. S é um conjunto de literais vigiados de c com uma valoração parcial P se qualquer uma das seguintes propriedades em S são verdadeiras:*

1. *possible*$(S, P) \geq max_{x_i \in S}(a + i)$

2. *current*$(S, P) \geq 0$

Proposição 2. *Se S é um conjunto de literais vigiados de c em uma valoração parcial P então c não é unitário.*

2.4.3 Aprendizado

O objetivo dos métodos de aprendizado é capturar a causa dos conflitos na forma de novas restrições aprendidas. A cláusula aprendida pode, então, ser adicionada ao conjunto de restrições para evitar que o mesmo conjunto de valorações ruins sejam feitas novamente. A implementação óbvia para o aprendizado em pseudo-Boolean é substituir a regra resolução em CNF com uma análoga em pseudo-Boolean.

Exemplo 3. *Suponha que possuímos a seguinte valoração parcial $P = \{c, e, \overline{b}, \overline{d}\}$, e as restrições:*

$$a + d + e \geq 2 \tag{2.3}$$

$$\overline{a} + b + c \geq 2 \tag{2.4}$$

Essas restrições causam à variável a ser forçada simultaneamente em 1 e 0. Durante a inspeção podemos identificar que esse conflito é causado por $\{\overline{b}, \overline{d}\}$ já que as valorações $\{c, e\}$ ajudam em satisfazer as restrições. O resolvente pseudo-Boolean para (2.3) e (2.4)

$$d + e + b + c \geq 3 \tag{2.5}$$

elimina as valorações ruins já que implica a cláusula $b \vee d$.

A restrição (2.5 faz mais do que eliminar o conflito imediato. Ela também elimina valorações ruins adicionais. Considere o equivalente lógico em cláusulas CNF

$$d \vee e \quad e \vee b$$

$$d \vee b \quad e \vee c$$

$$d \vee c \quad b \vee c$$

A restrição (2.5) também elimina $\{\bar{c}, \bar{e}\}$ que não é parte da valoração parcial atual, em um sentido de eliminar um erro que ainda não foi cometido.

2.5 Considerações finais

Ao longo deste capítulo vimos os principais algoritmos utilizados para resolver problemas SAT, Programação Inteira Mista e pseudo-Boolean. Embora em programação inteira o resolvedor trata a otimização de uma função objetivo, diferentemente de SAT e em muitos casos pseudo-Boolean, já que este último também pode ser visto como um problema de otimização, existe uma árvore de busca que é explorada e diversas técnicas utilizadas são implementadas a fim de explorar este espaço de forma mais rápida. Hoje é impossível identificar um resolvedor moderno com bons resultados sem a implementação de estruturas otimizadas para efetuar um BCP rápido, principalmente sem um sistema de aprendizado de conflitos que permita um retrocesso não-cronológico.

Todas essas estratégias implementadas nos resolvedores, causaram um enorme impacto no tempo e tamanho das fórmulas resolvidas. Cada estratégia implementada separadamente agrega alguma melhoria no tempo de processamento, mas a implementação de todas as estratégias - BCP rápido; aprendizado de cláusulas; retrocesso não cronológico e; heurística VSIDS na decisão - unidas apresentam o grande potencial de cada uma delas.

A implementação de todas as regras reflete em um resolvedor mais rápido e com capacidade de resolver problemas complexos em tempos aceitáveis, como exemplo de desempenho, um simples DPLL consegue resolver fórmulas, aleatórias, de até 100 variáveis e 430 cláusulas ao passo que o mais moderno resolvedor consegue resolver fórmulas de

domínios reais com mais de 1 milhão de variáveis e mais de 5 milhões de cláusulas.

Os avanços feitos em SAT podem ser, muitas vezes, aproveitados na implementação de resolvedores pseudo-Booleanos, como foi explicado no capítulo.

CAPÍTULO 3

REVISÃO DOS MÉTODOS DE PRÉ-PROCESSAMENTO

Um dos maiores avanços dos resolvedores SAT se dá pela incorporação do retrocesso não-cronológico e do BCP rápido, fazendo com que o resolvedor analise as cláusulas menos vezes. O retrocesso não-cronológico permite ao resolvedor identificar o conflito mais relevante e voltar diversas decisões até a problemática. Além disso o resolvedor é capaz de gerar novas cláusulas aprendidas, ajudando assim, a evitar que se pratique os mesmos erros de valoração que já cometeu.

Apesar dos grandes avanços citados acima, um momento chave para a solução de problemas SAT foi o pré-processamento das fórmulas, antes de se passar ao resolvedor. Algumas fórmulas possuem estruturas que podem dificultar para os resolvedores com essas técnicas modernas, mas que podem ser removidas ou simplificadas aplicando algumas regras. O pré-processador SAT que obteve maior sucesso foi o SatElite [25] que aplicou algumas técnicas nas fórmulas e com isso melhorou o tempo de solução dos resolvedores nas fórmulas pré-processadas.

Além dos pré-processadores SAT, a comunidade de pesquisa operacional também buscou métodos de pré-processamento para auxiliar o tempo de otimização dos problemas. A busca por pré-processamento em programação linear teve muitos trabalhos publicados nos anos de 1980 e 1990. Alguns trabalhos relevantes e que serão comentados ainda neste capítulo são: [30, 38, 15, 8, 19, 33, 58].

Mais recentemente a comunidade de SAT voltou a se preocupar com o pré-processamento. Ela tem trabalhado em técnicas que focam, também, nos problemas de MaxSAT. Em especial o artigo de Anton Belov [5], *SAT-based Preprocessing for MaxSAT* onde é afirmado que, o pré-processamento de fórmulas, em MaxSAT, é crucial para acelerar a solução SAT, já que os resolvedores no estado da arte de MaxSAT são uma iteração de chamadas em resolvedores SAT. Ainda nesse trabalho é defendida a idéia de que a fórmula deve ser

processada antes de ser enviada a um resolvedor ao invés (ou em adição) de a fazer entre as chamadas para o resolvedor SAT.

Em pseudo-Boolean também tivemos um trabalho importante de Dixon [24, 23, 22] onde houve uma preocupação com o pré-processamento das fórmulas, de onde saiu o fortalecimento de restrições, que foi uma releitura do trabalho da comunidade de pesquisa operacional. Mais recentemente, em 2009, Ruben Martins e outros publicaram o artigo *Preprocessing in Pseudo-Boolean Optimization: An Experimental Evaluation* [50], onde é feita uma releitura das ténicas de pré-processamento em SAT aplicando em fórmulas pseudo-Booleanas, e obtiveram um certo impacto positivo mas não tão grande quanto nas fórmulas em CNF.

A seguir dividimos o capítulo em três seções, uma explicando o pré-processamento em SAT, outra seção em programação linear mista e outra ainda sobre fortalecimento de restrições pseudo-Booleanas por fim fechamos o capítulo com as considerações.

3.1 Pré-processamento para SAT

As explicações nesta seção sobre o pré-processamento em SAT foram retiradas do trabalho de Anton Belov et. al [5], que compilou de forma esclarecedora o assunto.

Dada uma fórmula CNF F, o objetivo do pré-processamento para resolver problemas SAT é computar uma fórmula F' que seja equisatisfatível a F, e que deve ser mais fácil de ser resolvida. A computação de F' e do modelo de F obtido de F' no caso de $F' \in$ SAT, é esperado que seja rápida o suficiente para valer a pena todo o processo de resolver a fórmula. Muitas técnicas de pré-processamento de SAT dependem da combinação de pré-processamento baseado em resolução e eliminação de cláusulas. O pré-processamento baseado em resolução depende da aplicação da regra de resolução para *modificar* as cláusulas da fórmula de entrada e/ou reduzir o tamanho total da fórmula. Os procedimentos de remoção de cláusulas, por outro lado, não modificam as cláusulas da fórmula de entrada, apenas removem algumas das cláusulas, produzindo uma sub-fórmula da fórmula de entrada. As técnicas de pré-processamento em SAT podem ser descritas como procedimentos não determinísticos que aplicam passos atômicos na fórmula, inici-

almente de entrada, até um ponto fixo, ou até que recursos se esgotem, por exemplo o tempo.

O SatElite [25] é um dos pré-processadores SAT mais utilizados e que obteve grande sucesso na comunidade. As técnicas utilizadas pelo SatElite são: Eliminação de Variável limitada, do inglês *bounded variable elimination (BVE)*; eliminação por subjugação, resolução de auto-subjugado, do inglês *self-subsuming resolution (SSR)* e; propagação unitária. Adicionalmente uma técnica que é relevante na prática é a eliminação de cláusulas bloqueadas, do inglês *blocked clause elimination (BCE)* [36].

Eliminação de Variável limitada (BVE) [25] é um método de pré-processamento baseado em resolução, enraizada no algoritmo original de Davis-Putnam para SAT. Dada duas cláusulas $C_1 = (x \vee A)$ e $C_2 = (\neg x \vee B)$ o resolvente $C_1 \otimes_x C_2$ é a cláusula $(A \vee B)$. Para dois conjuntos F_x e $F_{\neg x}$ de cláusulas que possuam o literal x e $\neg x$, respectivamente, define $F_x \otimes_x F_{\neg x} = \{C_1 \otimes_x C_2 | C_1 \in F_x, C_2 \in F_{\neg x}, \text{ e } C_1 \otimes_x C_2 \text{ não é uma tautologia}\}$. A fórmula $ve(F, x) = F \setminus (F_x \cup F_{\neg x}) \cup (F_x \otimes_x F_{\neg x})$ é equisatisfatível com F, no entanto, em geral, pode ser quadrático no tamanho de F. Assim a operação atômica de eliminação de variável limitada é definida por $bve(F, x) = \text{ se } (|ve(F, x)| < |F|) \text{ então } ve(F, x) \text{ senão } F$. A fórmula $BVE(F)$ é obtida aplicando $bve(F, x)$ em todas as variáveis de F, lembrando que as implementações impõem restrições no BVE.

Eliminação por subjugação (SE) é um exemplo de técnica de eliminação de cláusulas. A cláusula C_1 subjuga a cláusula C_2, se $C_1 \subset C_2$. Para $C_1, C_2 \in F$, define $sub(F, C_1, C_2) = \text{ se } (C_1 \subset C_2) \text{ então } F \backslash \{C_2\} \text{ senão } F$. A fórmula $SUB(F)$ é obtida aplicando $sub(F, C_1, C_2)$ em todas as cláusula de F.

Propagação Unitária (UP) de uma cláusula unitária $(l) \in F$ é apenas a aplicação de $sub(F, (l), C)$ até um ponto fixo (para remover as cláusulas satisfeitas), seguido por $bve(F, var(l))$ (para remover a cláusula (l) e o literal $\neg l$ das cláusulas remanescentes).

Resolução de auto-subjugado (SSR) utiliza resolução e eliminação por subjugação. Dada duas cláusulas $(C_1 = (l \vee A)$ e $C_2 = (\neg l \vee B)$ em F, de foma que $A \subset B$, temos que $C_1 \otimes_l C_2 = B \subset C_2$, e então C_2 pode ser substituído por B ou, em outras palavras, $\neg l$ é removido de C_2. Consequentemente, o passo atômico SSR, $ssr(F, C_1, C_2)$, resulta na

fórmula $F \setminus \{C_2\} \cup \{B\}$ se C_1, C_2 são como acima, e $F,$ caso contrário.

O passo atômico da *eliminação de cláusulas bloqueadas (BCE)* consiste em remover uma cláusula bloqueada — uma cláusula $C \in F$ é dita bloqueada em F [41], se para algum literal $l \in C$, todo resolvente de C com $C' \in F$ em l é tautológico. Uma fórmula $BCE(F)$ é obtida aplicando $bce(F, C) =$ se (C bloquada em F) então $F \setminus \{C\}$ senão F para todas as cláusulas de F. Note que uma cláusula com um literal puro é bloqueada, então a eliminação de literal puro é um caso especial de BCE. BCE possui uma propriedade importante chamada *monotoniciedade*: para qualquer $F' \subseteq F, BCE(F') \subseteq BCE(F)$. É mantido porque C é bloqueado para F, e também será bloqueado para qualquer subconjunto de F. A eliminação por subjugação não é monotônica.

3.2 Pré-processamento para Programação Inteira Mista

As explicações nesta seção sobre o pré-processamento em programação inteira mista foram retiradas do trabalho de Savelsbergh [58], que compilou de forma esclarecedora o assunto.

Na programação inteira, as técnicas de pré-processamento visam reduzir o tamanho dos coeficientes na matriz de restrições, para reduzir os limites das variáveis. Nesta seção exploraremos as técnicas que foram abordadas na criação do *pBCR*.

As técnicas que abordaremos são clássicas e utilizadas pelo MINTO [58], um framework de pré-processamento de fórmulas para programação linear inteira mista, e também explorados pelo moderno resolvedor SCIP [1], são a de pré-processamento básico, técnicas básicas de sondagem, implicações lógicas, técnica aprimorada de sondagem e desigualdades do tipo clique.

Para esta seção, assumiremos que a desigualdade em consideração é considerada na seguinte forma:

$$\sum_{j \in B^+} a^i_j x_j - \sum_{j \in B^-} a^i_j x_j + \sum_{j \in C^+} g^i_j y_j - \sum_{j \in C^-} g^i_j y_j \leq b,$$

onde $B = B^+ \cup B^-$ é o conjunto de variáveis binárias, $C = C^+ \cup C^-$ é o conjunto de variáveis inteiras e contínuas, e $a_j > 0$ para $j \in B$ e $G_j > 0$ para $j \in C$. Não faremos

distinção entre variáveis inteiras e contínuas. Além disso, assumiremos que o limites superiores e inferiores das variáveis inteiras e contínuas serão denotadas por l_j e u_j.

3.2.1 Técnicas Básicas de Pré-processamento

Identificação de inviabilidade, considere o seguinte problema de programação inteira mista

$$z \; = \; min \sum_{j \in B^+} a^i_j x_j - \sum_{j \in B^-} a^i_j x_j + \sum_{j \in C^+} g^i_j y_j - \sum_{j \in C^-} g^i_j y_j$$

sujeito a

$$A^i x + G^i y \leq b^i$$

$$l \leq y \leq u$$

$$x \in \{0,1\}^n, y \in RR^m$$

Se $z > b_i$, então obviamente a região de viabilidade é vazia. Infelizmente, o problema de programação linear mista acima é tão difícil de ser resolvido quanto o problema original, mas deve ficar claro que qualquer limite inferior de z satisfaz $(z \geq z_{LB} > b_i)$. O limite inferior mais simples, e também o mais fraco, é obtido por descartar completamente o sistema $A^i x + G^i y \leq b^i$. Neste caso podemos concluir que o sistema é inviável se

$$- \sum_{j \in B^-} a^i_j + \sum_{j \in C^+} g^i_j l_j - \sum_{j \in C^-} g^i_j u_j > b_i$$

Identificação de Redundância, considerando o seguinte problema de programação in-

teira mista

$$z \;=\; max \sum_{j \in B^+} a^i_j x_j - \sum_{j \in B^-} a^i_j x_j + \sum_{j \in C^+} g^i_j y_j - \sum_{j \in C^-} g^i_j y_j$$

sujeito a

$$A^i x + G^i y \leq b^i$$

$$l \leq y \leq u$$

$$x \in \{0,1\}^n, y \in RR^m$$

Se $z \leq b_i$, então a desigualdade $a^i x + g^i y \leq b_i$ é redundante. O problema acima é tão difícil de se resolver quanto o problema original, mas deve ficar claro que qualquer limite superior de z satisfaz ($z \leq z_{UB} \leq b_i$). O limite superior mais simples, e também o mais fraco, é obtido descartando o sistema $A^i x + G^i y \leq b^i$. Neste caso, podemos concluir que uma desigualdade é redundante se

$$\sum_{j \in B^+} a^i_j + \sum_{j \in C^+} g^i_j u_j - \sum_{j \in C^-} g^i_j l_j \leq b_i$$

3.2.2 Técnicas de Sondagem Básica

As técnicas de sondagem são baseadas na investigação de consequências lógicas, i.e, definir provisoriamente uma variável binária x_k para 0 ou 1 e explorar as consequências.

Considere uma variável binária $x_k, k \in B^+$ e a seguinte extensão do problema de programação linear mista

$$z_k \;=\; min \sum_{j\in B^+} a^i_j x_j - \sum_{j\in B^-} a^i_j x_j + \sum_{j\in C^+} g^i_j y_j - \sum_{j\in C^-} g^i_j y_j$$

sujeito a

$$A^i x + G^i y \leq b^i$$

$$x_k = 1$$

$$l \leq y \leq u$$

$$x \in \{0,1\}^n, y \in RR^m$$

Neste caso $x_k > b_i$, então a região plausível nesta formulação estendida é vazia. Portanto, $x_k \neq 1$ em qualquer solução plausível para a formulação original e x_k pode ser fixado em 0. Quando descartamos o sistema $A^i x + G^i y \leq b^i$, podemos fixar a variável x_k em 0 se

$$a^i_j - \sum_{j\in B^-} a^i_j + \sum_{j\in C^+} g^i_j l_j - \sum_{j\in C^-} g^i_j u_j > b_i$$

Em seguida, consideramos a variável $x_k, k \in B^-$ e a seguinte formulação estendida do problema

$$z_k \;=\; min \sum_{j\in B^+} a^i_j x_j - \sum_{j\in B^-} a^i_j x_j + \sum_{j\in C^+} g^i_j y_j - \sum_{j\in C^-} g^i_j y_j$$

sujeito a

$$A^i x + G^i y \leq b^i$$

$$x_k = 0$$

$$l \leq y \leq u$$

$$x \in \{0,1\}^n, y \in RR^m$$

Se $z_k > b_i$, então a região plausível para esta formulação estendida é vazia. Portanto, $x_k \neq 0$ em qualquer solução plausível na formulação original e x_k pode ser fixada em 1.

Quando descartamos o sistema $A^i x + G^i y \leq b^i$, podemos fixar a variável x_k em 1 se

$$- \sum_{j \in B^- \setminus \{k\}} a_j^i x_j + \sum_{j \in C^+} g_j^i l_j - \sum_{j \in C^-} g_j^i u_j > b_i$$

3.2.3 Implicações Lógicas

As técnicas básicas de pré-processamento e sondagem podem ser utilizadas para derivar implicações lógicas entre as variáveis.

Primeiramente, considere a variável binária $x_{k_1}, k_1 \in B$ e uma variável contínua $y_k, k \in C$ e a seguinte versão estendida do problema de programação linear mista

$$z_{k_1}^{i_1} = min \sum_{j \in B^+} a_j^{i_1} x_j - \sum_{j \in B^-} a_j^{i_1} x_j + \sum_{j \in C^+ \setminus \{k\}} g_j^{i_1} y_j - \sum_{j \in C^-} g_j^{i_1} y_j$$

sujeito a

$$A^{i_1} x + G^{i_1} y \leq b^{i_1}$$

$$x_{k_1} = 1$$

$$l \leq y \leq u$$

$$x \in \{0,1\}^n, y \in RR^m$$

e

$$z_{k_2}^{i_2} = min \sum_{j \in B^+} a_j^{i_2} x_j - \sum_{j \in B^-} a_j^{i_2} x_j + \sum_{j \in C^+ \setminus \{k\}} g_j^{i_2} y_j - \sum_{j \in C^-} g_j^{i_2} y_j$$

sujeito a

$$A^{i_2} x + G^{i_2} y \leq b^{i_2}$$

$$x_{k_2} = 1$$

$$l \leq y \leq u$$

$$x \in \{0,1\}^n, y \in RR^m$$

Temos que, $y_k \leq (b_{i_1} - z_{k_1}^{i_1})/g_k^{i_1}$ e $y_k \geq (z_{k_1}^{i_2} - b_{i_2})/g_k^{i_2}$, ou seja, a análise da versão esten-

dida do sistema de desigualdades pode revelar que quando $x_{k_1} = 1$, os limites superiores e inferiores na variável y_k podem ser melhorados. No caso dos limites melhorados fixarem a variável y_k, i.e., $l_k = u_k = v_k$, teremos estabelecido a implicação lógica $x_{k_1} = 1 \Rightarrow y_k = v_k$. Analogamente, pode ser possível estabelecer a implicação lógica $x_{k_1} = 0 \Rightarrow y_k = v_k$.

Considerando duas variáveis binárias $x_{k_1}, k_1 \in B$ e $x_{k_2}, k_2 \in B$ e a seguinte formulação estendida

$$z \ = \ min \sum_{j \in B^+} a^i_j x_j - \sum_{j \in B^-} a^i_j x_j + \sum_{j \in C^+} g^i_j y_j - \sum_{j \in C^-} g^i_j y_j$$

sujeito a

$$A^i x + G^i y \leq b^i$$

$$x_{k_1} = 1$$

$$x_{k_2} = 1$$

$$l \leq y \leq u$$

$$x \in \{0,1\}^n, y \in RR^m$$

Se $z > b_i$, então a região plausível para a formulação estendida é vazia. Neste caso, as seguintes implicações lógicas são identificadas

$$x_{k_1} = 1 \Rightarrow x_{k_2} = 0,$$

$$x_{k_2} = 1 \Rightarrow x_{k_1} = 0.$$

De maneira similar, as seguintes implicações lógicas podem ser identificadas

$$x_{k_1} = 0 \Rightarrow x_{k_2} = 0, \qquad x_{k_2} = 0 \Rightarrow x_{k_1} = 0$$

$$x_{k_1} = 0 \Rightarrow x_{k_2} = 1, \qquad x_{k_2} = 0 \Rightarrow x_{k_1} = 1$$

$$x_{k_1} = 1 \Rightarrow x_{k_2} = 1, \qquad x_{k_2} = 1 \Rightarrow x_{k_1} = 1.$$

2

3.2.4 Técnica Aprimorada de Sondagem

As técnicas básicas de sondagem fixam provisoriamente uma variável binária para algum de seus limites e exploram as consequências lógicas. Suponha, que sem perda de generalidade, que a variável x_k é provisoriamente fixada em seu limite superior. Se já existe alguma implicação lógica associada à variável x_k sendo fixada em seu limite superior, ela pode ser feita durante a busca por outra consequência lógica. É importante observar que a efetuação de uma implicação lógica pode ser o gatilho para várias outras implicações lógicas.

Exemplo

$$\min\ 24x_1 + 12x_2 + 16x_3 + 4y_1 + 2y_2 + 3y_3$$

sujeito a

$$y_1 + 3y_2 \geq 15$$

$$y_1 + 2y_3 \geq 10$$

$$2y_1 + y_2 \geq 20$$

$$y_1 \leq 15x_1$$

$$y_2 \leq 20x_2$$

$$y_3 \leq 5x_3$$

$$y_1, y_2, y_3 \in \mathbb{R}_+$$

$$x_1, x_2, x_3 \in \{0, 1\}$$

A aplicação da técnica básica de pré-processamento irá fixar os limites superiores das variáveis contínuas em 15, 20 e 5, respectivamente. A implicação lógica que pode ser identificada da formulação e suas derivações são listadas abaixo

Note que algumas destas implicações lógicas não seriam identificadas se os limites superiores das variáveis contínuas não pudessem ser melhoradas pelas técnicas básicas de

Implicação	Derivação
$x_1 = 0 \Rightarrow \quad y_1 = 0$	$(x_1 = 0 \Rightarrow y_1 = 0)$
$y_2 = 20$	$(x_1 = 0 \Rightarrow y_1 = 0 \Rightarrow y_2 = 20)$
$y_3 = 5$	$(x_1 = 0 \Rightarrow y_1 = 0 \Rightarrow y_3 = 5)$
$x_2 = 1$	$(x_1 = 0 \Rightarrow y_1 = 0 \Rightarrow y_2 = 20 \Rightarrow x_2 = 1)$
$x_3 = 1$	$(x_1 = 0 \Rightarrow y_1 = 0 \Rightarrow y_3 = 5 \Rightarrow x_3 = 1)$
$x_2 = 0 \Rightarrow \quad y_2 = 0$	$(x_2 = 0 \Rightarrow y_2 = 0)$
$y_1 = 15$	$(x_2 = 0 \Rightarrow y_2 = 0 \Rightarrow y_1 = 15)$
$x_1 = 1$	$(x_2 = 0 \Rightarrow y_2 = 0 \Rightarrow y_1 = 15 \Rightarrow x_1 = 1)$
$x_3 = 0 \Rightarrow \quad y_3 = 0$	$(x_3 = 0 \Rightarrow y_3 = 0)$
$x_1 = 1$	$(x_3 = 0 \Rightarrow y_3 = 0 \Rightarrow y_1 \geq 10 \Rightarrow x_1 = 1)$

pré-processamento. Também perceba que $y_3 = 0 \Rightarrow y_1 \geq 10$, embora, de acordo com a definição, não seja uma implicação lógica, é usada na derivação de $x_3 = 0 \Rightarrow x_1 = 1$.

A aplicação da técnica aprimorada de sondagem resultou na seguinte fórmula melhorada

$$\min \ 24x_1 + 12x_2 + 16x_3 + 4y_1 + 2y_2 + 3y_3$$

sujeito a

$$45x_1 + y_1 + 3y_2 \geq 60$$

$$5x_2 + y_1 + 2y_3 \geq 15$$

$$10x_2 + 2y_1 + y_2 \geq 30$$

$$y_1 \leq 15x_1$$

$$y_2 \leq 20x_2$$

$$y_3 \leq 5x_3$$

$$0 \leq y_1 \leq 15$$

$$0 \leq y_2 \leq 20$$

$$0 \leq y_3 \leq 5$$

$$x_1, x_2, x_3 \in \{0, 1\}$$

O valor da solução inicial da programação linear relaxada é 58.70. O valor da solução melhorada é 79.10. O valor da solução ótima é 79.33.

3.2.5 Desigualdades Clique

As implicações lógicas podem ser utilizadas para derivar desigualdades clique, i.e, desigualdades com apenas variáveis binárias na forma $\sum_{j \in S^+} x_j - \sum_{j \in S^-} x_j \leq 1 - |S^-|$. Desigualdades clique tem sua fundação teórica no trabalho de Johnson e Padberg [38]. A construção das desigualdades clique é baseada na implicação lógica entre as variáveis binárias. Existem quatro tipos de implicações lógicas entre variáveis binárias e podem ser representadas como abaixo:

- $x_i = 1 \Rightarrow x_j = 0$
- $x_i = 1 \Rightarrow \neg x_j = 0$
- $\neg x_i = 1 \Rightarrow x_j = 0$
- $\neg x_i = 1 \Rightarrow \neg x_j = 0$

onde $\neg x_k = 1 - x_k$ denota o complemento de x_k. Uma implicação lógica identifica duas variáveis, tanto originais como complementares, que não podem ser 1 ao mesmo tempo em qualquer solução plausível. Construir o grafo $G = (B^o \cup B^c, E)$ com B^o sendo o conjunto de variáveis binárias originais, B^c sendo o conjunto do complemento das variáveis binárias, e E o conjunto de arestas, onde duas variáveis são ligadas com uma aresta se e somente se essas duas variáveis não podem ser 1 ao mesmo tempo em uma solução factível. Consequentemente, cada implicação define uma aresta no grafo. Além disso, existe uma aresta para cada variável e seu complemento.

Não é difícil identificar que cada clique maximal $C = C^o \cup C^c$, com $C^o \subseteq B^o$ e $C^c \subseteq B^c$, definem uma desigualdade clique

$$\sum_{j \in C^o} x_j + \sum_{j \in C^c} \neg x_j \leq 1$$

implicando em

$$\sum_{j \in C^o} x_j - \sum_{j \in C^c} \neg x_j \leq 1 - |C^c|$$

Duas observações importante podem ser feitas a respeito das desigualdades clique:

- se $|C^o \cap C^c| = 1$ e $k \in C^o \cap C^c$, então $x_j = 0$ para todo $j \in C^o \setminus \{k\}$ e $x_j = 1$ para todo $j \in C^c \setminus \{k\}$.

- se $|C^o \cap C^c| > 1$ o problema é infactível.

Exemplo Considerando o seguinte sistema de desigualdades

$$4x_1 + x_2 - 3x_4 \leq 2$$

$$2x_1 + 3x_2 + 3x_4 \leq 7$$

$$x_1 + 4x_2 + 2x_3 \leq 5$$

$$3x_1 + x_2 + 5x_3 \leq 6$$

$$x_1, x_2, x_3, x_4 \in \{0, 1\}$$

temos as seguintes implicações identificadas

Implicação	Derivação
$x_1 = 1 \Rightarrow \quad x_4 = 1$	$(x_1 = 0 \Rightarrow x_4 = 1)$
$x_2 = 0$	$(x_1 = 1 \Rightarrow x_4 = 1 \Rightarrow x_2 = 0)$
$x_3 = 0$	$(x_1 = 1 \Rightarrow x_4 = 1 \Rightarrow x_3 = 0)$
$x_2 = 1 \Rightarrow \quad x_3 = 0$	$(x_2 = 1 \Rightarrow x_3 = 0)$
$x_3 = 1 \Rightarrow \quad x_2 = 0$	$(x_3 = 1 \Rightarrow x_2 = 0)$
$x_1 = 0$	$(x_3 = 1 \Rightarrow x_1 = 0$
$x_4 = 0 \Rightarrow \quad x_1 = 0$	$(x_4 = 0 \Rightarrow x_1 = 0)$

O grafo $G = (B^o \cup B^c, E)$ para o sistema de desigualdades é dado na figura 3.1 e contém apenas um clique maximal de cardinalidade maior que dois: $\{x_1, x_2, x_3\}$. Este clique define a desigualdade $x_1 + x_2 + x_3 \leq 1$.

3.3 Fortalecimento Pseudo-Booleano

O fortalecimento consiste em uma técnica, que constrói de forma automática de inferências. O objetivo de construir inferências é inferir novas restrições de um conjunto de restrições clausais. O efeito é construir uma representação mais forte e concisa de um problema, partindo de uma representação maior e mais fraca.

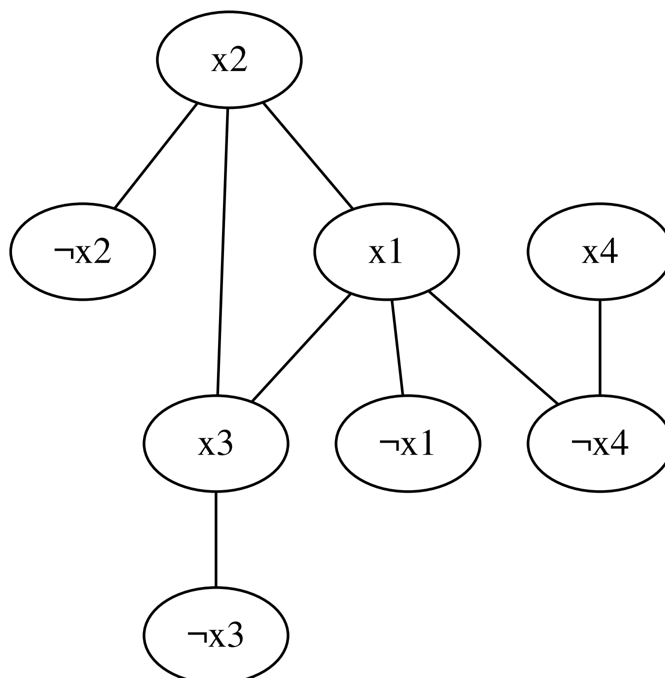

Figura 3.1: Grafo auxiliar para a formação da desigualdade clique

O método de fortalecimento é uma forma de redução de coeficientes. É uma técnica herdada da área de pesquisa operacional usada principalmente para pré-processamento de problemas inteiros mistos [24, 23, 22], mais especificamente o método de fortalecimento é uma adaptação da técnica aprimorada de sondagem, explicada na seção 3.2.4. O procedimento é uma adaptação de um DPLL padrão.

Definição 6. *Uma restrição c na forma $\sum a_i x_i \geq k$ é dita super satisfeita em uma valoração parcial P se $current(c, P) > 0$*

Entrada: Fórmula pseudo-Boolean C
Saída: Fórmula fortalecida
1 **para cada** *literal l em C* **faça**
2 $\quad P \leftarrow l$
3 $\quad P \leftarrow$ Propagação(C, P)
4 $\quad$ **para cada** *restrição $c = \sum a_i x_i \geq k$ com $current(c, P) > 0$* **faça**
5 $\quad\quad c \leftarrow \sum a_i x_i + current(c, P) \neg l \geq k + current(c, P)$
6 $\quad$ **fim**
7 **fim**

Algoritmo 3: Fortalecimento

O procedimento de fortalecimento é iniciado com $P = \emptyset$. Suponhamos que valoremos l, adicionando em P, e então aplicamos a propagação em nosso conjunto de restrições.

Agora descobrimos que nessa valoração parcial P, a restrição c está super satisfeita com $current(c, P) = s$ para algum $s > 0$. A restrição super satisfeita pode ser substituída pela versão fortalecida da restrição.

Proposição 3. *Dado um conjunto de restrições C e uma valoração $P = \{l\}$, se $P' =$ Propagação(C, P) , então qualquer restrição c na forma $\sum a_i x_i \geq r$ com $current(c, P') = s$ pode ser substituído pela restrição*

$$s\bar{l} + \sum a_i x_i \geq r + s \tag{3.14}$$

Exemplo 4. *Considere o seguinte conjunto de cláusulas:*

$$a + b \geq 1 \tag{3.15a}$$

$$a + c \geq 1 \tag{3.15b}$$

$$b + c \geq 1 \tag{3.15c}$$

Se tivermos $P = \{\neg a\}$, geraremos pela propagação $\{b, c\}$. A restrição 3.15c é super satisfeita e pode ser substituída pela restrição:

$$a + b + c \geq 2$$

Essa nova restrição subjuga todas as três restrições originais, então 3.15a e 3.15b podem ser removidas do conjunto de restrições. Uma restrição fortalecida geralmente irá subjugar algumas ou todas as restrições envolvidas em sua geração.

Essa regra pode ser generalizada como segue.

Proposição 4. *Dado um conjunto de restrições C, uma valoração parcial $P = \{l_1, l_2, \ldots, l_k\}$, valoração $P' =$ Propagação(C, P), e a restrição $c \in C$ na forma $\sum a_i x_i \geq r$ com $current(c, P') = s$ e $s > 0$, a restrição*

$$s \sum_{i=1}^{k} \neg l_i + \sum a_i x_i \geq r + s \tag{3.16}$$

é implicada por C.

3.4 Considerações Finais

Neste capítulo fizemos uma revisão das principais técnicas de pré-processamento utilizado pela comunidade de SAT, técnicas para pré-processamento de programação linear mista e também o fortalecimento para fórmulas pseudo-Booleanas que é derivada das técnicas de programação linear mista.

CAPÍTULO 4

MODELO PROPOSTO

Os capítulos anteriores revisaram o estado da arte no quesito de decisão dos problemas SAT e pseudo-Boolean. Abordaram as principais técnicas utilizadas em Pesquisa Operacional para pré-processar os problemas a fim de ter tempos de decisão e otimização menores do que sem qualquer pré-processamento.

A comunidade de pseudo-Boolean ainda é bastante recente no meio do grupo de Satisfatibilidade. É interessante notar que, embora pseudo-Boolean seja muito próxima de SAT, pouco das técnicas de pré-processamento foram trazidas para os resolvedores SAT atuais. Existem alguns estudos que aplicam técnicas de pré-processadores SAT, o que não é o foco desta tese.

Neste capítulo apresentamos um novo processo de pré-processamento pseudo-Booleano que utiliza, em sua maioria, técnicas trazidas da área de Pesquisa Operacional. Batizamos este processo de *pseudo-Boolean Constraint Reduction (pBCR)*.

O *pBCR* é um processo de pré-processamento, ou seja, ele não vai resolver a fórmula, apenas vai gerar uma nova instância que seja equi-satisfatível, na expectativa, de que seja uma fórmula mais fácil (que leve menos tempo) para o resolvedor. Dividimos o *pBCR* em quatro etapas:

1. *Sondagem* - Etapa de sondagem dos literais da fórmula, onde é feita a verificação das inferências com a fixação de valores de algumas variáveis;

2. *Aprendizado* - Etapa de aprendizado, onde novas restrições são injetadas na fórmula;

3. *Fortalecimento* - Etapa de fortalecimento das restrições;

4. *Subjugação* - Etapa de subjugação de restrições.

Todos os processos, acima resumidos, derivam de estratégias abordadas em diversos campos da resolução de problemas conforme explicado no capítulo 3.

A figura 4.1 ilustra onde o $pBCR$ se encaixa durante o processo de resolver uma dada fórmula como entrada.

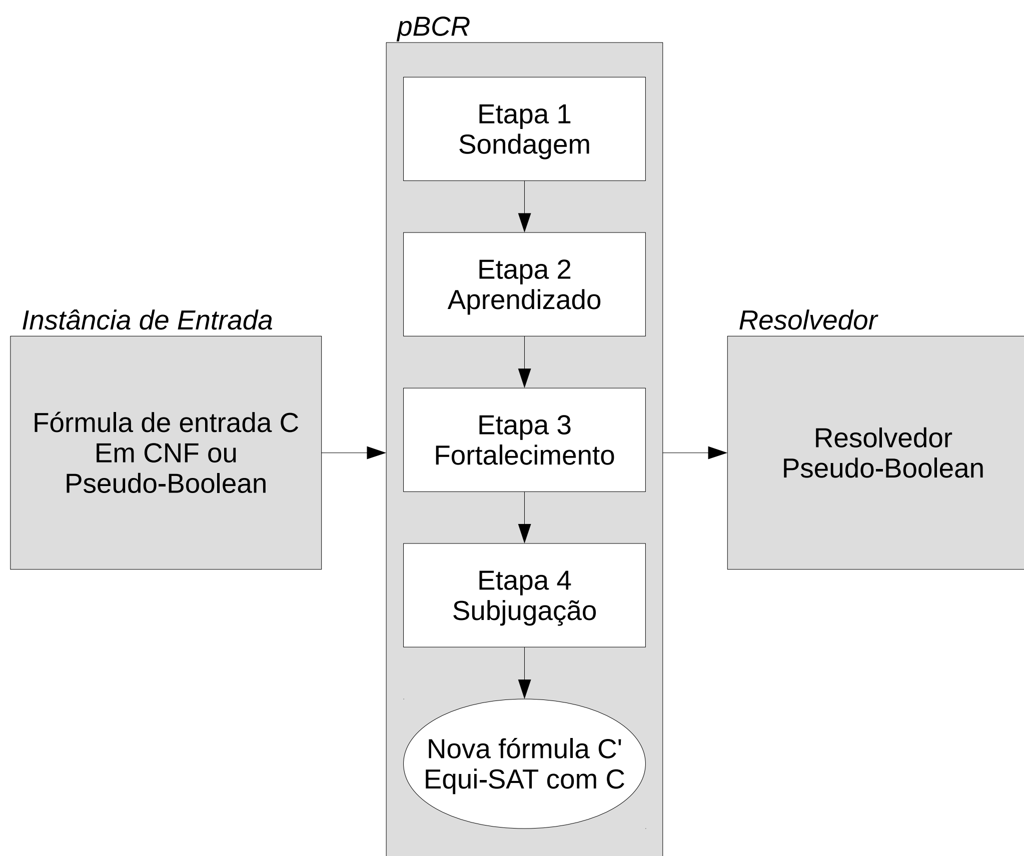

Figura 4.1: Diagrama do processo de aplicação do pré-processador $pBCR$ em uma fórmula de entrada

Este capítulo está dividido em seções para cada uma das etapas: Sondagem; Aprendizado; Fortalecimento; e Subjugação de Restrições. E é finalizado com considerações.

4.1 Sondagem

A técnica de sondagem consiste na investigação das consequências lógicas, isto é, fixar uma variável qualquer x_k para *Verdadeiro* ou *Falso* e explorar as consequências lógicas da propagação.

Esta técnica é bastante cara computacionalmente pois, a princípio, todos os literais devem ser investigados, levando à investigação de $2 * V$, onde é V é a quantidade de

variáveis na fórmulas, já que cada um dos lados, o literal original ou negado, devem ser testados de uma variável.

Para evitar que se gaste muito tempo na sondagem algumas técnicas foram criadas afim de explorar um conjunto menor de variáveis.

A sondagem implementada no *pBCR* observa alguns fatores antes de começar. Primeiramente são identificados os literais que fazem parte de cláusulas unitárias ou restrições pseudo-Boolean decididas.

Definição 7. *Uma restrição pseudo-Boolean decidida é uma restrição em que a soma dos coeficientes associados aos literais seja igual ao valor do lado direito da restrição. Neste caso todos os literais devem ser valorados como verdadeiro para que se possa satisfazer a restrição em questão.*

Após a identificação desses fatores, o algoritmo sorteia mais algumas variáveis para preencher a quantidade mínima de variáveis a serem exploradas na sondagem. O parâmetro desta quantidade pode ser modificado, mas o algoritmo por padrão inicia o processo com 5% das variáveis existentes na fórmula, e em todas essas variáveis elas serão sondadas para *Verdadeiro* e *Falso*.

Durante o processo, os literais que fazem parte de restrições que foram eliminadas com alguma decisão são adicionados ao processo de sondagem para explorar suas consequências lógicas.

Além disso, a sondagem também faz eliminação de variáveis que devem ser valoradas em algum de seus lados, senão causará a insatisfatibilidade de alguma restrição. Neste caso esta variável recebe a valoração e, sua implicação lógica é propagada na fórmula de forma que valore outras variáveis e remova restrições.

Quando uma variável causa conflito em qualquer uma de suas valorações, a fórmula é dita como trivialmente insatisfatível. Para a sondagem determinar uma fórmula como trivialmente insatisfatível, podem ocorrer em dois casos: O primeiro é quando uma variável causa conflito em qualquer uma de suas valorações, mas esse tipo de ocorrência é bastante raro; e o segundo caso ocorre quando o conjunto de propagações força a valoração de algumas variáveis e suas inferências, conflitam com as inferências de outras decisões.

Vejamos um exemplo simples que determina o acionamento da insatisfatibilidade trivial:

Para o caso base é a fórmula:

$$+1x_1 \geq 1$$
$$+1\neg x_1 \geq 1$$

No caso exemplificado acima a fórmula é insatisfatível pois não importa qual valor x_1 tome, alguma restrição sempre ficará falsa.

Quando a sondagem termina de passar por todas variáveis adicionadas e nenhuma outra foi adicionada para exploração, o processo acaba e passa para a próxima fase. Desta maneira o processo pode, em um caso extremo, explorar todas as variáveis da fórmula de entrada.

O algoritmo 4 descreve o funcionamento da sondagem. Inicia pela escolha de alguns literais para começar a sondagem. Em seguida percorre a lista dos literais a serem sondados e os coloca como valoração parcial, conforme identificado na linha 4 do algoritmo. Então é executada a função de propagação. Caso a propagação retorne conflito, é feita uma nova propagação com a valoração parcial, sendo apenas o oposto do literal que causou conflito. Havendo conflito novamente a fórmula é dita insatisfatível. Quando não há o conflito, a valoração é fixada na fórmula. Caso não ocorra conflito na primeira tentativa, o algoritmo procura, pelos literais não valorados e não sondados, das restrições que foram satisfeitas com a decisão, e os coloca como literais a serem sondados. Acrescendo, deste modo, a quantidade de literais a serem sondados pelo algoritmo, afim de aumentar a gama de literais trabalhados no algoritmo. Além disso, se o oposto do literal sendo sondado no momento não tiver sido sondado em sua negação, ele é adicionado à lista de literais a serem sondados.

Entrada: Fórmula pseudo-Boolean C
Saída: Uma nova fórmula C' contida em C

```
1  C′ ← C
2  JaSondada ← Ø
3  Asondar ← EscolheConjuntoMinimoparaSondar()
4  para cada literal l ∈ Asondar, l ∉ JaSondada faça
5      P ← l
6      P ← Propagação(C′,P)
7      JaSondada ← JaSondada ∪ l
8      se Conflito() então
9          Propagação(C′, ¬l)
10         se Conflito() então
11             retorna UNSAT
12         senão
13             C′ ← FixaValoração(C, ¬l)
14     senão
15         para cada restrição c ∈ restrições satisfeitas por P faça
16             para cada literal n ∈ c, n ≠ l, n ∉ JaSondada faça
17                 Asondar ← Asondar ∪ n
18         se ¬l ∉ JaSondada, ¬l ∉ Asondar então
19             Asondar ← Asondar ∪ ¬l
20 retorna C′
```

Algoritmo 4: Algoritmo da sondagem

4.2 Aprendizado

O processo de aprendizagem é a etapa do *pBCR* que aproveita as implicações decorrentes da exploração das consequências lógicas e insere lemas na nova fórmula de saída do *pBCR*.

A inclusão de lemas durante o processo de busca foi um dos grandes avanços dos resolvedores de SAT. Elas evitam o resolvedor de incorrer nos mesmos erros de valoração passados no processo de busca. Por isso, a etapa de aprendizado do *pBCR* é bastante importante, pois tira proveito das implicações decorrentes dos testes na etapa de sondagem, afim de ajudar o resolvedor a evitar caminhos que serão, eventualmente, descartados pelo resolvedor com seu maquinário, com a tendência de evitar retrocessos e redescobertas já feitas pelo pré-processamento.

Ao longo do processo de sondagem, todas as implicações geradas pelas decisões exploratórias são armazenadas. Para uma variável x_k sabe-se quais implicações são geradas pela decisão x_k e $\neg x_k$.

Essas implicações são atribuídas a x_k mesmo que uma variável $x_{k'}$ implicada por x_k implique em $x_{k''}$. Podemos dizer que por transitividade que $x_{k''}$ é uma implicação da decisão de x_k já que $x_{k'}$ é implicação direta de x_k e qualquer outra implicação somente acontece com a decisão x_k, pois no momento da sondagem apenas a variável em questão foi valorada, toda e qualquer outra valoração acontece por implicação direta desta valoração iniciada na sondagem.

Logo armazenamos o seguinte conjunto de implicações:

Para uma decisão de sondagem dizemos que I é o conjunto de variáveis implicadas por esta decisão, então:

$$x_k \rightarrow \bigwedge_{i \in I} x_i \tag{4.2}$$

Note que a implicação não vale na volta, já que não necessariamente uma decisão em qualquer uma das variáveis implicadas por x_k implicam em x_k.

Além da implicação gerada por cada decisão de sondagem também são armazenadas as decisões que implicam um literal, ou seja, para um literal x_l armazenamos todas as

decisões da etapa de sondagem que implicam em x_l, ficando:

$$\bigvee_{i \in L} x_i \rightarrow x_l \tag{4.3}$$

Neste caso temos que para um literal qualquer x_l temos o conjunto de decisões que implicam em sua valoração. É importante notar que esse conjunto é guardado como um $\vee$ (ou), pois não é necessário que todas as decisões sejam feitas para que este literal seja valorado desta forma, basta que apenas um deles seja valorado para acontecer tal literal.

Por isso, o processo armazena quais decisões implicam em x_l e $\neg x_l$, e com esses conjuntos armazenados podemos inserir as restrições que podem ajudar o resolvedor a não tentar alguns conjuntos de valorações durante o processo de busca.

É trivial identificar que os literais x_l e $\neg x_l$ não podem ocorrer ao mesmo tempo em um modelo, pois de x_l for verdadeiro logo $\neg x_l$ é falso. Então utilizamos essa variável como âncora do que aprendemos durante a sondagem.

Como já definido em [38, 58] criamos um grafo onde cada literal não pode ser valorado em conjunto, então o literal x_l não pode ser verdadeiro juntamente com os literais que implicam em $\neg x_l$ já que x_l não pode ocorrer com $\neg x_l$.

Ainda em [38, 58] podemos adicionar como restrições os sub-grafos completos, ficando cada literal do sub-grafo completo como um termo da restrição e a desigualdade ficando menor ou igual a 1,representando que somente um dos literais podem ser marcados como verdadeiro para que não aconteça conflito na fórmula, ficando então uma restrição no formato $\sum x_i \leq 1$.

Essa abordagem é bastante válida quando se procura os sub-grafos completos maximais dentre todos os conflitos, porém esta busca é muito custosa computacionalmente, ficando inviável e por isso trabalhamos com uma versão simplificada desta ideia.

Os cliques maximais neste grafo reduzido que estamos trabalhando são pequenos (em geral apenas 2 vértices) e bastante numerosos. Criar todas as restrições pequenas fica inviável para o resolvedor já que a quantidade de restrições cresce em uma escala exponencial, e por isso reduzimos para no máximo 2 restrições extras por variável.

Podemos identificar que a restrição pode ser feita com todos os literais ligados ao literal em questão ficando o peso 1 para todos os literais e para o literal em questão o peso sendo a quantidade de literais que não podem ocorrer ao mesmo tempo, ficando as restrições:

$$\sum_{i \in I} x_i + |I| \neg x_l \leq |I| \tag{4.4}$$

$$\sum_{i \in I'} x_i + |I'| x_l \leq |I'| \tag{4.5}$$

Para exemplificar, sabemos que para a variável x_{72} temos as seguintes inferências:

$$x_{99} \vee x_{100} \vee x_{101} \rightarrow x_{72}$$

$$\neg x_{31} \vee x_{46} \vee \neg x_{56} \vee x_{76} \rightarrow \neg x_{72}$$

Logo podemos adicionar as seguintes restrições:

$$+1x_{99} + 1x_{100} + 1x_{101} + 3\neg x_{72} \leq 3$$

$$+1\neg x_{31} + 1x_{46} + 1\neg x_{56} + 1x_{76} + 4x_{72} \leq 4$$

O algoritmo 5 descreve o funcionamento deste processo de aprendizado. O algoritmo inicia percorrendo todos os literais da fórmula que são implicados por alguma decisão, conforme a linha 1. Então para cada um desses literais, o algoritmo identifica todas os literais t_i que implicam o literal l, definidos no conjunto $I[l]$, conforme a linha 3, então a nova restrição é construída conforme definido acima.

```
   Entrada: Conjunto de Implicados por variável I; Lista de literais da fórmula L
   Saída: Conjunto de Restrições aprendidas A
 1 para cada literal l ∈ L faça
 2 |   c ← ∅
 3 |   para todo literal t_i ∈ I[l] faça
 4 |   |_ c ← ∑ +1 t_i  + |I[l]|¬l ≤ |I[l]|
 5 |_  A ← A + c
 6 retorna A
```

Algoritmo 5: Algoritmo inserir restrições aprendidas na fórmula

Quando algum dos lados da variável x_k são implicados por apenas 1 literal podemos escrever apenas uma restrição, como descrita no algoritmo 6 que é apenas um refinamento do algoritmo 5. Neste refinamento, ao invés de se adicionar duas restrições, apenas uma restrição será adicionada e esta não conterá o literal de x_k mas sim o literal que implica x_k, como no exemplo abaixo:

$$x_9 \vee x_{10} \vee x_{11} \rightarrow x_2$$
$$x_6 \rightarrow \neg x_2$$

Logo podemos adicionar a seguinte restrição:

$$+1x_{99} + 1x_{100} + 1x_{101} + 3x_6 \leq 3$$

No algoritmo 6 identificamos nas linhas 3 a 6 que se algum conjunto de implicados $I[l]$ ou $I[\neg l]$ possuem tamanho igual a 1, então é formada uma restrição com os literais do maior conjunto somado ao literal pertencente ao conjunto de tamanho 1, como descrito na linha 5. E por fim, na linha 6, a negação do literal l é removida do conjunto de literais a serem percorridos para adicionar restrições aprendidas.

Embora as restrições nos algoritmos 5 e 6 não estejam na forma normalizada, o *pBCR* injeta na fórmula essas restrições em sua forma normalizada e posiform. Apenas deixamos desta forma para melhor entendimento do leitor.

4.2.1 Aprendizado Avançado

Além do processo padrão de aprendizada implementado no *pBCR*, temos um outro método de aprendizagem que gera restrições mais fortes, porém com uma complexidade computacional mais elevada. Esta maneira de aprendizagem é uma generalização da busca dos cliques maximais.

Com as implicações devidamente armazenadas agora podemos adicionar as novas res-

Entrada: Conjunto de Implicados por variável I; Lista de literais da fórmula L
Saída: Conjunto de Restrições aprendidas A

```
1  para cada literal l ∈ L faça
2      c ← ∅
3      se |I[l]| = 1 ∨ |I[¬l]| = 1 então
4          para todo literal t_i ∈ max(I[l], I[¬l] faça
5              c ← ∑ +1 t_i + max(|I[l]|, |I[¬l]|) t ∈ min(I[l], I[¬l]) ≤
               max(|I[l]|, |I[¬l]|)
6              L ← L \ ¬l
7      senão
8          para todo literal t_i ∈ I[l] faça
9              c ← ∑ +1 t_i + |I[l]|¬l ≤ |I[l]|
10     A ← A + c
11 retorna A
```

Algoritmo 6: Algoritmo inserir restrições aprendidas na fórmula, aprimoramento do algoritmo 5

trições na fórmula. As novas restrições são uma composição das implicações de x_k e $\neg x_k$, onde sabemos que todas as implicações de x_k não podem ocorrer em conjunto com as restrições de $\neg x_k$. Para cada variável temos dois conjuntos de implicações, sendo eles:

$$x_k \rightarrow \bigwedge_{i \in I} x_i \tag{4.10}$$

$$\neg x_k \rightarrow \bigwedge_{i \in I'} x_i \tag{4.11}$$

Para cada literal decidido na etapa da sondagem é possível construir uma restrição contendo os literais que não podem ocorrer simultaneamente. Criando, então, um conjunto de restrição mais forte que o método mais simplificado de aprendizado, explicado na seção anterior.

O algoritmo escolhe um literal e busca os literais que conflitam com as inferências deste primeiro literal. Quando o literal tem suas inferências conflitantes com a do literal inicial, ele é adicionado no grafo e ligado com todos os literais que conflitam com ele. Quando nenhum outro literal puder ser adicionado ao grafo o processo termina e a nova

restrição é gerada. A nova restrição é escrita no formato abaixo:

$$\sum \delta(x_i)x_i \leq max(\delta_x) \tag{4.12}$$

Esta restrição é inspirada na restrição do clique maximal como um caso especial da restrição acima, conforme a conjectura abaixo.

Conjectura 1. *A restrição gerada pelo clique maximal que diz que a soma dos literais conflitantes entre si deve ser menor ou igual a 1 equivale à equação 4.12, pois sendo um grafo completo o grau de todos os nós será o mesmo e por isso a constante inteira pode ser simplificada para 1.*

A conjectura sendo provada verdadeira será possível gerar restrições possuindo variáveis que não fazem parte do clique maximal.

Para exemplificar o processo podemos tomar como exemplo de implicações os seguintes literais:

$$x_{70} \rightarrow x_{71} \wedge x_{76} \wedge x_{100} \tag{4.13a}$$

$$x_{140} \rightarrow \neg x_{71} \wedge x_{180} \wedge x_{15} \tag{4.13b}$$

$$x_{141} \rightarrow \neg x_{71} \wedge x_{142} \tag{4.13c}$$

$$x_{142} \rightarrow \neg x_{71} \wedge \neg x_{180} \tag{4.13d}$$

$$x_{143} \rightarrow \neg x_{71} \wedge \neg x_{80} \wedge \neg x_{142} \tag{4.13e}$$

Para seguir o exemplo vamos tomar o literal x_{70} como o literal inicial em questão para a construção da restrição, que tem suas implicações definidas na equação 4.13a. Durante a busca do algoritmo podemos identificar que o literal x_{140} conflita com x_{70} pois uma de suas implicações é o literal $\neg x_{71}$, logo os literais x_{70} e x_{140} não podem ocorrer ao mesmo tempo em um modelo, por isso o nosso grafo será representado pela ligação entre esses dois literais, conforme a figura 4.2a.

Continuando o algoritmo, identificamos que o literal x_{141} também conflita com o literal x_{70} por causa da implicação $\neg x_{71}$. Neste ponto adicionamos x_{141} ao grafo conectado a x_{70}

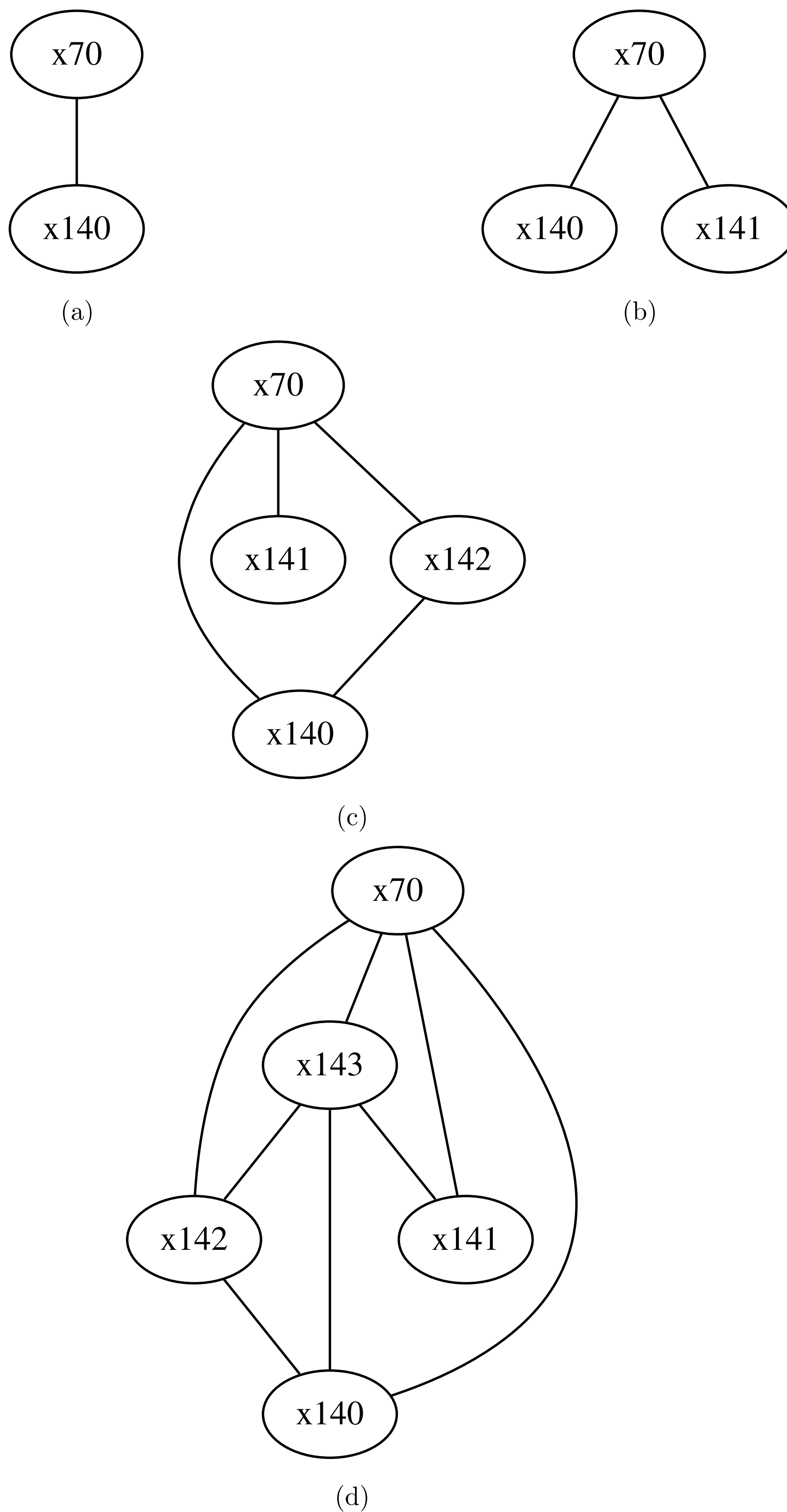

Figura 4.2: Etapas de construção do grafo de Aprendizado Avançado, para o exemplo

e verificamos se x_{141} e os outros nós, no caso x_{140}, conflitam entre si. Não sendo o caso, o grafo fica como representado na figura 4.2b, e passamos para o próximo literal.

Agora temos o literal x_{142} que conflita com x_{70} pela inferência $\neg x_{71}$. Adicionamos ao grafo ligado a x_{70} e verificamos se existe algum outro conflito com outro literal. Neste caso, acontece o conflito com o literal x_{140} por causa da variável x_{180}, onde, cada um tem uma inferência oposta. Então estes dois literais devem ser ligados, tendo o grafo resultante como o da figura 4.2c.

Finalmente, em nosso exemplo, o último literal a conflitar com x_{70} é o x_{143}, também pela inferência de $\neg x_{71}$. Na busca por conflitos mútuos, identificamos que o último literal adicionado conflita com todos os outros já adicionados ao grafo. Com esta última adição, o grafo fica finalizado como representado na figura 4.2d.

Por fim, seguindo a equação 4.12 a nova restrição a ser adicionada à fórmula é:

$$+4x_{70} + 4x_{143} + 3x_{140} + 2x_{141} + 3x_{142} \leq 4 \tag{4.14}$$

Embora tenhamos este método de aprendizado definido e conjecturado, a sua implementação não foi feita no *pBCR*.

4.3 Fortalecimento

O fortalecimento no *pBCR* acontece da mesma forma do que já foi explicado na seção 3.3, que é uma releitura de Dixon nos pré-processamentos da pesquisa operacional que abordamos na seção 3.2.4 e descrito em profundidade no trabalho de Savelsbergh [58]. Porém, o processo não é, necessariamente, feito com todos os literais da fórmula.

Aqui os literais que serão verificados como possíveis literais a serem fortalecidos são os mesmo utilizados durante o processo da sondagem. Se não forem literais decididos eles serão reaplicados na fórmula para identificar alguma restrição super-satisfeita.

O algoritmo 7 descreve como funciona o processo de fortalecimento no *pBCR*. A função de fortalecimento recebe como entrada o conjunto de literais a serem explorados, que foram utilizados na etapa de sondagem, e também recebe o conjunto de restrições da fórmula

C. Para cada literal no conjunto L o algoritmo marca como valoração parcial P o literal em questão como verdadeiro, em seguida executa a propagação e guarda as restrições que foram satisfeitas com esta propagação. Então para cada restrição satisfeita é verificada se ela está supersatisfeita, ou seja, se a soma dos coeficientes dos literais valorados excede o lado direito da desigualdade, se $current(c, P) > 0$. Com a restrição supersatisfeita o literal l é adicionado em sua negação com coeficiente sendo o valor de $curret(c, P)$, o valor da restrição é acrescido a neste mesmo valor.

Entrada: Lista de literais a serem explorados L
Conjunto de Restrições da fórmula C
Saída: Conjunto de Restrições fortalecidas F

```
1 F ← ∅
2 para cada literal l ∈ L faça
3     P ← l
4     RestriçõesSatisfeitas ← Propagação(C,P)
5     para cada restrição c ∈ RestriçõesSatisfeitas faça
6         se c tem current(c, P) > 0 então
7             c ← ∑ a_i x_i + current(c, P)¬l ≥ k + current(c, P)
8             F ← F ∪ c
9 retorna F
```

Algoritmo 7: Algoritmo para fortalecer restrições

4.4 Subjugação de Restrições

A subjugação acontece logo após a etapa do fortalecimento onde restrições mais fortes podem subjugar um conjunto de restrições que fizeram parte de sua formação mais forte.

O processo de subjugação é bastante caro em fórmulas pseudo-Boolean, já que uma restrição pode subjugar outra quando todos os modelos da restrição C_i também são modelos da restrição C. Quando as restrições em questão são cláusulas basta que uma cláusula C_i seja um subconjunto de outra cláusula C_j ($C_i \subseteq C_j$), então $C_i \models C_j$, porém em restrições pseudo-Booleanas isso não é sempre verdade já que uma restrição pode precisar

de um subconjunto de literais para tornar-se verdadeira como nas restrições:

$$a + b \geq 1 \tag{4.15a}$$

$$a + b + c \geq 2 \tag{4.15b}$$

A restrição 4.15b subjuga a restrição 4.15a pois em todos os seus modelos deverá possuir os literais a ou b valorados como verdadeiro, e na restrição 4.15a basta que somente um desses literais sejam valorados como verdadeiro. Mas a restrição 4.15a não subjuga a restrição 4.15b pois nem todos os seus modelos satisfazem a restrição 4.15b

O custo computacional para identificar as restrições subjugadas torna essa aplicação inviável, mesmo em fórmulas pequenas. Porém conseguimos identificar que ao longo do processo de fortalecimento um fenômeno acontece com as restrições, facilitando, assim, o processo de subjugação, conforme exemplificado abaixo.

O processo de subjugação ocorre de forma natural, mesmo que não em sua totalidade, após o fortalecimento pois conforme o progresso do fortalecimento as restrições que fizeram parte do fortalecimento de uma restrição são fortalecidas da mesma maneira, de forma que no fim do processo as restrições que seriam subjugadas tornam-se iguais às restrições que seriam as subjugadoras.

Para exemplificar melhor, tomemos as seguintes restrições:

$$a + b \geq 1 \tag{4.16a}$$

$$a + c \geq 1 \tag{4.16b}$$

$$b + c \geq 1 \tag{4.16c}$$

- Se $P = \{a\}, \{\}$
- Se $P = \{\neg a\}$, $\{b, c\}$ será propagado

- A restrição 4.16c é super satisfeita e pode ser substituída pela restrição:

$$a + b + c \geq 2$$

- Se $P = \{b\}, \{\}$
- Se $P = \{\neg b\}$, $\{a, c\}$ será propagado
- A restrição 4.16b é super satisfeita e pode ser substituída pela restrição:

$$a + b + c \geq 2$$

- Se $P = \{c\}, \{\}$
- Se $P = \{\neg c\}$, $\{a, b\}$ será propagado
- A restrição 4.16a é super satisfeita e pode ser substituída pela restrição:

$$a + b + c \geq 2$$

- Duas restrições podem ser removidas da fórmula.
- A restrição 4.16a é super satisfeita e pode ser substituída pela restrição:

$$a + b + c \geq 2$$

Ficamos com a fórmula:

$$a + b + c \geq 2 \tag{4.17a}$$

$$a + b + c \geq 2 \tag{4.17b}$$

$$a + b + c \geq 2 \tag{4.17c}$$

logo, podemos remover as restrições 4.17b e 4.17c da fórmula, ficando apenas a restrição 4.17a.

Com estas afirmações podemos formular uma nova conjectura.

Conjectura 2. *Para as subjugações de restrições em uma fórmula pseudo-Booleana C fortalecida, o conjunto de restrições fortalecidas C_F é suficiente para identificar as restrições a serem subjugadas. Se uma restrição c_f é fortalecida pelas inferências geradas por um conjunto $C_{FI} \subseteq C$, sabemos que c_f irá subjugar um subconjunto $C_{FI'} \subseteq C_{FI}$. Para evitar que se faça a verificação em cada restrição, temos que as restrições em C_{FI} subjugáveis por c_f também serão fortalecidas pelas inferências da sondagem de $C_{FI'} \cup c_f$, ficando representadas da mesma maneira que a restrição c_f, podendo então, ser feita a subjugação apenas pela identificação das restrições igualmente representadas.*

De acordo com a conjectura acima, as restrições que seriam subjugadas e , portanto, removidas da fórmula crescem de tal forma que ficam iguais às restrições subjugadoras. Isto pode não ocorrer em todos os casos, mas pelos experimentos, acontecem de forma suficiente a reduzir a quantidade de restrições na fórmula e, possivelmente, auxiliando o resolvedor a melhorar o seu tempo de solução.

```
   Entrada: Conjunto de Restrições da fórmula C;
            Conjunto de Restrições fortalecidas F
   Saída: Restrições subjugadas
1  L ← Ø
2  para cada restrição c em F faça
3      se c ∈ L então
4          C ← C \ c
5      senão
6          L ← L ∪ c
```

Algoritmo 8: Algoritmo para encontrar e subjugar restrições fortalecidas

A implementação do sistema de subjugação é bastante simples. Basta armazenar quais são as restrições na fórmula que foram fortalecidas durante o processo de fortalecimento. Após o fortalecimento começamos com uma lista vazia L e a lista de restrições fortalecidas F. Para cada restrição fortalecida identificamos se uma restrição igual está na lista de

restrições L, se estiver então a restrição em questão deverá ser subjugada, caso contrário, deverá ser adicionada à lista L, conforme é descrito no algoritmo 8.

4.5 Considerações Finais

Neste capítulo foram apresentadas as técnicas utilizadas pela comunidade de pesquisa operacional aplicadas em fórmulas da comunidade de satisfatibilidade.

A aplicação deste conjunto de técnicas traz uma nova maneira da comunidade de satisfatibilidade interpretar as fórmulas pseudo-Boolean, tanto em suas versões de otimização como decisão, já que apenas havia sido agregada a técnica de fortalecimento, trazida por Heidi Dixon em seus trabalhos. Em SAT muito se discute em pré-processamento de fórmulas, inclusive no capítulo 3 discutimos a existência de técnicas aplicadas em SAT, mas pouco se discute a aplicação de técnicas em fórmulas pseudo-Boolean, ou ainda aplicar técnicas em fórmulas SAT resultando em uma nova fórmula pseudo-Boolean, como trataremos no capítulo seguinte.

Podemos dizer que o momento de explorar técnicas de pré-processamento em fórmulas pseudo-Boolena é bastante atual e se encaixa dentro da comunidade, já que diversos grupos estão explorando técnicas de pré-processamento de fórmulas SAT e MaxSAT, como acontece em trabalhos recentes [25, 50, 5].

O *pBCR* faz o pré-processamento da fórmula, de forma a tentar reduzir o tempo de resolução dos resolvedores modernos e para isso implementou técnicas que estão presentes nos resolvedores modernos, como: a propagação rápida, por meio de literais vigiados; injeção de lemas na fórmula por meio de técnicas baseadas no aprendizado; e fortalecimento de restrições. Também discute uma nova maneira de injetar restrições na fórmula como uma generalização da técnica clássica de sondagem avançada, explicada na seção 3.2.4.

Durante o desenvolvimento do *pBCR*, duas conjecturas foram feitas: uma relativa ao aprendizado avançado, e; outra relativa à subjugação de restrições após o fortalecimento.

A conjectura 1, referente ao aprendizado avançado, se provada, mostra que é possível ter uma generalização do aprendizado por desigualdades clique.

A conjectura 2, referente à subjugação, mostra que é possível fazer subjugação das restrições que involvem os fortalecimentos sem a necessidade de verificar se todos os modelos de uma restrição são os modelos da outra, causando a enumeração das tabelas verdades das restrições envolvidas.

No próximo capítulo faremos um estudo empírico da aplicação do $pBCR$ em diversas fórmulas com algumas configurações de teste com as etapas do algoritmo.

CAPÍTULO 5

AVALIAÇÃO EXPERIMENTAL

Neste capítulo apresentamos os experimentos executados com o *pBCR* para validar cada uma das quatro etapas.

Os resolvedores escolhidos para rodar os experimentos são:

- para fórmulas em CNF:
  - LINGELING [6];
  - GLUCOSE [63, 3]; e
  - CLASP [29].
- para fórmulas em pseudo-Boolean:
  - CLASP; e
  - SCIP [1].

O LINGELING e GLUCOSE foram escolhidos baseados em suas classificações nas competições anteriores de SAT e em suas recentes técnicas, como a simplificação de fórmulas em seu pré-processamento, conforme abordado no capítulo 3.

O resolvedor CLASP foi escolhido por ser o melhor resolvedor pseudo-Boolean disponível até o momento da redação desta tese. Além do CLASP foram avaliados:

- o Sat4J-PB, que é uma variação do SAT4J [42] para pseudo-Boolean;
- BSOLO [45], que foi descartado por ser disponibilizado apenas em binário 32bits, não conseguindo resolver as maiores instâncias avaliadas; e
- WBO [46] incorre no mesmo problema do BSOLO.

O resolvedor de programação linear SCIP foi escolhido por estar entre os melhores resolvedores de programação linear gratuitos e livres, e também por efetuar diversas técnicas

de pré-processamento abordadas nesta tese, como a sondagem e a injeção restrições aprendidas baseadas nos cliques, como definido na seção 3.2.5.

O ambiente de teste foi em um Intel Xeon E5-2690 de 3GHz com 256GB de memória RAM. O tempo limite de execução por resolvedor foi de 3.600 segundos para as fórmulas de decisão (ambas CNF e pseudo-Booleanas) e de 1.800 segundos para as fórmulas de otimização pseudo-Booleanas. Cada resolvedor teve acesso a 22 GB de memória RAM.

As fórmulas escolhidas para a execução dos testes são oriundas de diversos domínios de planejamento e, de competições de SAT e pseudo-Boolean de anos anteriores.

Os domínios de planejamento são do tipo *strips* ou "planejamento clássico". Os usados em nossa avaliação experimental são Domínio dos Blocos, Childsnack, Depots, Logistics, Driverlog, Elevator, FreeCell, Gripper, Satellite, Storage, Thoughtful e TPP.

Esses domínios são das competições de planejamento de 2014 e anteriores[1]. Este conjunto de experimento gerou 764 fórmulas CNF. As fórmulas foram extraídas do SATPLAN [39] nos passos em que gerava a fórmula e passava para o resolvedor CNF.

Além das fórmulas oriundas de problemas de planejamento, temos as fórmulas da SAT-Race 2008 [61], totalizando 49 fórmulas, e mais 23 fórmulas de verificação de processadores de Miroslav e colegas [66], totalizando 836 fórmulas todas em CNF.

Para as fórmulas exclusivamente pseudo-Boolean utilizamos as fórmulas da competição de pseudo-Boolean de 2012, totalizando 116 fórmulas para decisão (sem otimização) e 650 fórmulas para otimização.

No total, 1.602 fórmulas fizeram parte da avaliação experimental desta tese. Dividimos os experimentos em vários pedaços que exploram as particularidades de cada uma das etapas do *pBCR*.

5.1 Teste da Sondagem

Para experimentar somente a sondagem, que é a etapa 1 do *pBCR*, desligamos as etapas 2,3 e 4, conforme representado na figura 5.1.

[1] As competições de planejamento podem ser acessadas pelo site do ICAPS, uma importante conferência sobre planejamento: http://icaps-conference.org/index.php/Main/Competitions

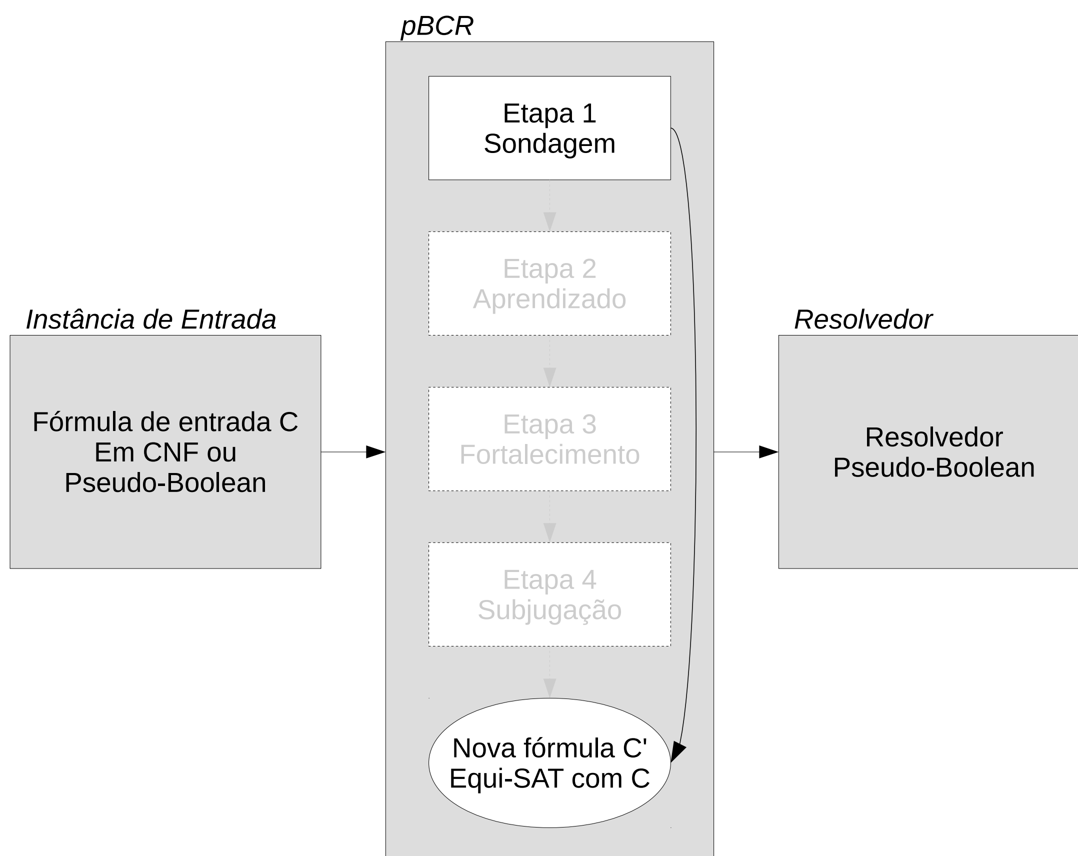

Figura 5.1: Diagrama do processo de aplicação do pré-processador *pBCR* no experimento da Sondagem, etapa 1

Neste experimento, é interessante notar, que se a fórmula de entrada é uma CNF, a saída continua sendo CNF, já que o processo de sondagem não modifica as restrições agregando informação, apenas removendo restrições e diminuindo literais em cada restrição.

Para executar os experimentos separamos em duas etapas, a primeira etapa executamos o *pBCR* apenas com fórmulas CNF e a segunda etapa apenas com fórmulas pseudo-Booleanas de decisão e, posteriormente, apenas com fórmulas de otimização. Esta seção está divida em subseções considerando cada uma das etapas citadas.

5.1.1 Fórmulas CNF

Em nossa bancada de experimentos contamos com 823 fórmulas exclusivamente em CNF. E executamos o *pBCR* com apenas a sondagem ligada, conforme a figura 5.1 e geramos fórmulas CNF.

Dentre estas fórmulas, sabemos que 764 são provenientes de problemas de planejamento geradas pelo SATPLAN, cada uma, representando um estágio de sua busca no grafo de planos. E com isso para cada problema temos diversas fórmulas sendo que somente uma será satisfatível. As primeiras fórmulas são, geralmente, trivialmente insatisfatíveis. E pudemos observar que o *pBCR* conseguiu identificar essas fórmulas apenas com sua etapa de sondagem.

Fórmulas Trivialmente Insatisfatíveis

Conforme discutido na seção 4.1 existem casos em que fórmulas trivialmente insatisfatíveis podem ser identificadas. E foi o ocorrido: das 764 fórmulas de planejamento, 278 foram identificadas como trivialmente insatisfatíveis pelo *pBCR*. Além destas fórmulas, mais 9 fórmulas do benchmark de Miroslav e colegas também foram identificadas. Totalizando 287 fórmulas identificadas como trivialmente insatisfatíveis.

Apesar destas fórmulas serem triviais, ainda demandam um tempo de processamento para serem identificadas, e com isso comparamos o tempo que o *pBCR* demorou com os resolvedores LINGELING, GLUCOSE e o CLASP.

O *pBCR*, com tempo combinado, identificou todas essas fórmulas em 10.816 segundos. O LINGELING em 2.010 segundos. O GLUCOSE em 1.898 segundos. O CLASP em CNF levou 4.724 segundos.

Embora o *pBCR* tenha obtido um tempo total superior aos resolvedores utilizados, 76% do tempo total destas 287 fórmulas está concentrado em somente 10 fórmulas com tempos que superam 400 segundos e 2 fórmulas que superam 1.000 segundos. Enquanto que no LINGELING, as 10 fórmulas que levam mais tempo representam 82% do tempo total, somando 1.665 segundos. No CLASP, apenas uma fórmula consumiu 3.084 segundos de processamento, representando 65% do tempo total de processamento e apenas 2 fórmulas consumiram mais de 400 segundos. O GLUCOSE possui 1 fórmula que ultrapassa 400 segundos, consumindo 814 segundos de processamento, o que representa 42% do tempo total.

Dadas estas diferenças é difícil identificar se, em um âmbito geral, os resolvedores e o

pBCR ficam próximos em suas soluções. Por isso aplicamos o teste de Friedman com o pós-teste de Nemenyi para identificar se existe alguma diferença estatística nos tempos. No apêndice A explicamos como funciona este teste. Neste teste estatístico classificamos cada um dos resolvedores em cada uma das fórmulas, gerando uma classificação média de acordo com a tabela 5.1. As médias das classificações são muito próximas para identificar diferenças sem utilizar algum método estatístico. Por isso o teste de Friedman foi executado, retornando o valor F $204,46$, identificando, assim, diferença estatística entre os resolvedores. Após executar o pós-teste de Nemenyi pudemos gerar um gráfico com a diferença crítica, conforme a figura 5.2. Nesta figura podemos identificar que o LINGELING e GLUCOSE não possuem diferença estatística, assim como o GLUCOSE e o *pBCR*, e o último também não possui diferença estatística com o CLASP.

O *pBCR* mostrou-se bastante competitivo em sua implementação, já que implementa a propagação com literais vigiados. A justificativa de possuir um tempo global maior está no fato do *pBCR* não prosseguir com a valoração, pois a cada decisão feita, a anterior é esquecida, enquanto que os resolvedores sempre ampliam a valoração parcial, ficando mais fácil de encontrar conflitos.

Resolvedor	Classificação Média
LINGELING	1,524100
GLUCOSE	1,672764
pBCR	1,880662
CLASP	1,950348

Tabela 5.1: Classificação Média dos resolvedores no tempo utilizado para decidir que a fórmula em questão é insatisfatível.

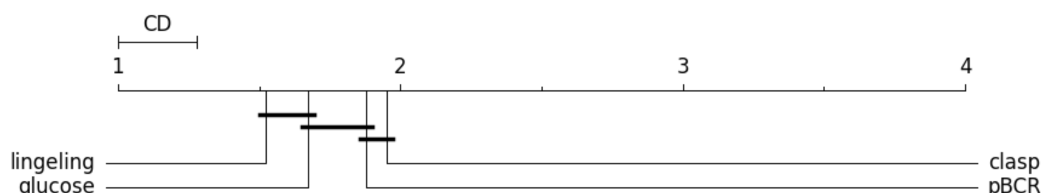

Figura 5.2: Comparação estatística do pós-teste de Nemenyi

Fórmulas Não Trivialmente Insatisfatíveis e Satisfatíveis

Apenas um pequeno conjunto de fórmulas foi identificado como trivialmente insatisfatível dentre as fórmulas CNF. Agora vamos discutir como foi o resultado da sondagem para todas as outras fórmulas restantes, que são 549.

O *pBCR* levou $7.719,35$ segundos para executar a sondagem nestas 549 fórmulas. Destas, 12 fórmulas levam mais de 100 segundos para serem sondadas e representam $58,75\%$ do tempo total de sondagem do conjunto das 549 fórmulas, e uma fórmula utilizou $1.119,93$ segundos para a sondagem.

Neste experimento dividimos a análise por resolvedor, para identificar os efeitos da modificações nas fórmulas.

LINGELING

Dentre as 549 fórmulas o LINGELING não conseguiu decidir 11 fórmulas no tempo limite estipulado, e são 10 fórmulas que não são resolvidas com o LINGELING dentre as fórmulas sondadas. Uma das fórmulas que tiveram o tempo limite excedido foi resolvida pelo LINGELING quando sondada. Esta fórmula foi resolvida em $207,6$ segundos pelo LINGELING e demorou $7,7$ segundos para ser sondada.

Nas 538 fórmulas que foram resolvidas pelo LINGELING em ambas configurações, original e sondada, temos $35.060,6$ segundos de tempo para as fórmulas originais e $30.940,2$ segundos nas fórmulas sondadas. O *pBCR* demorou $7.719,35$ segundos para sondar estas fórmulas e por isso o LINGELING com o *pBCR* demoram $38.659,55$ segundos. A redução de tempo foi de $11,75\%$ quando não se considera o tempo de sondagem, e há um acréscimo de 10% em relação à execução nas fórmulas originais quando se considera o tempo que o *pBCR* utiliza na sondagem.

Na figura 5.3 cada ponto representa uma fórmula, e estão representadas as 538 fórmulas que foram decididas dentro do tempo limite de execução, do LINGELING nas fórmulas originais, e nas fórmulas sondadas pelo *pBCR*. O tempo utilizado pelo LINGELING nas fórmulas sondadas foi indexado pelo eixo x, enquanto que nas fórmulas originais foi indexado no eixo y. A linha $f(x) = x$ foi traçada como referência para auxiliar na identificação das fórmulas que são melhores quando sondadas e originais. Todos os pontos acima da

linha representam que a fórmula sondada teve tempo inferior ao das fórmulas originais. Neste gráfico foi considerado o tempo utilizado pela sondagem.

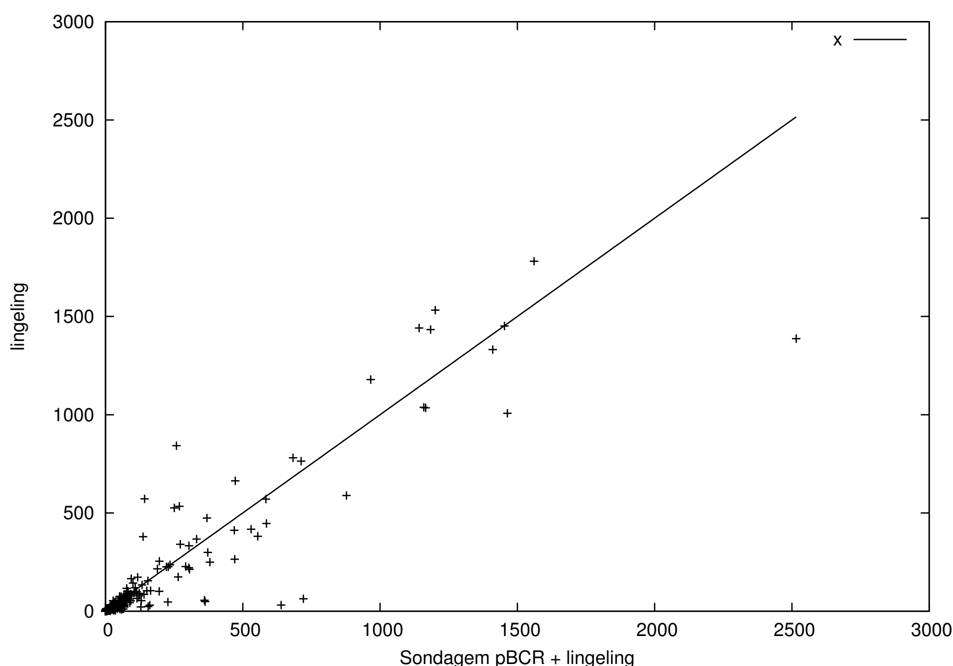

Figura 5.3: Comparação dos tempos de execução do LINGELING nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do LINGELING no eixo y contra o LINGELING com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência.

Dentre todas as 538 fórmulas o LINGELING em cima das versões sondadas, e considerando o tempo do *pBCR*, obteve um menor tempo em 311 fórmulas perdendo em 227. Também é importante notar que 1 (uma) fórmula só pôde ser resolvida, com o limite de tempo estipulado, após sua sondagem.

CLASP

Considerando o resolvedor CLASP, temos 17 fórmulas das 549 com o limite de tempo excedido, já nas fórmulas sondadas foram 13 fórmulas com o tempo limite excedido, ou seja, nas fórmulas sondadas o CLASP teve 4 fórmulas em vantagem, se comparado com as fórmulas originais. Das fórmulas que se diferem,em uma delas, o tempo limite foi excedido quando executado na fórmula sondada, mas na fórmula original o CLASP utilizou $955,87$ segundos de processamento. Doze fórmulas tiveram o tempo limite excedido em ambas configurações, e 5 fórmulas foram resolvidas em sua versão sondada enquanto que

ultrapassavam o tempo limite na original.

Das fórmulas que o CLASP resolveu a mais na versão de sondagem, tivemos os tempos sumarizados na tabela 5.2 abaixo:

CLASP	Sondagem	Total
419,51	22,33	441,84
1.002,44	57,22	1.059,66
1.099,86	66,26	1.166,12
3.071,96	217,99	3.289,95
62,17	1.119,94	1.182,10

Tabela 5.2: Tempo de execução em segundos do resolvedor CLASP nas fórmulas sondadas que foram resolvidas mas que tiveram o tempo limite excedido em suas configurações originais.

Quando se considerou o conjunto das configurações, o CLASP executando nas fórmulas originais e nas sondadas, tivemos 531 fórmulas que foram resolvidas mutuamente nas configurações. Destas fórmulas o tempo total gasto pelo CLASP nas fórmulas originais foi de $36.403,3$ segundos e de $24.501,6$ segundos em cima das fórmulas sondadas, causando uma redução de $32,69\%$ no tempo de solução das fórmulas. Quando se considera o tempo do $pBCR$ executando a sondagem, o tempo total, i.e, tempo do CLASP nas fórmulas mais o tempo do $pBCR$ nas fórmulas tivemos $30.402,2$ segundos, tendo o $pBCR$ levado $5.900,6$ segundos de processamento para a sondagem. Considerando o tempo do $pBCR$, o CLASP teve uma redução de $16,48\%$ de tempo em relação a execução das fórmulas originais.

Dentre as 531 fórmulas mutuamente decididas nas configurações, sem considerar o tempo utilizado pelo $pBCR$, o CLASP obteve o menor tempo de processamento em 392 fórmulas, mas o número é reduzido para 129 quando se considera o tempo utilizado para processar a sondagem nas fórmulas. A figura 5.4 mostra a comparação dos tempos de decisão das fórmulas, tendo no eixo x os tempos pelo CLASP com a sondagem e no eixo y o tempo do CLASP nas fórmulas originais, quando os pontos estiverem acima da linha $f(x) = x$, então o tempo da fórmula sondada foi inferior em relação à fórmula original.

Estes resultados com o CLASP foram obtidos utilizando-se fórmulas representadas em CNF, que também são aceitas pelo CLASP. Mas quando se escreve as fórmulas, tanto as sondadas como as originais, em formato pseudo-Boolean os resultados obtidos pelo

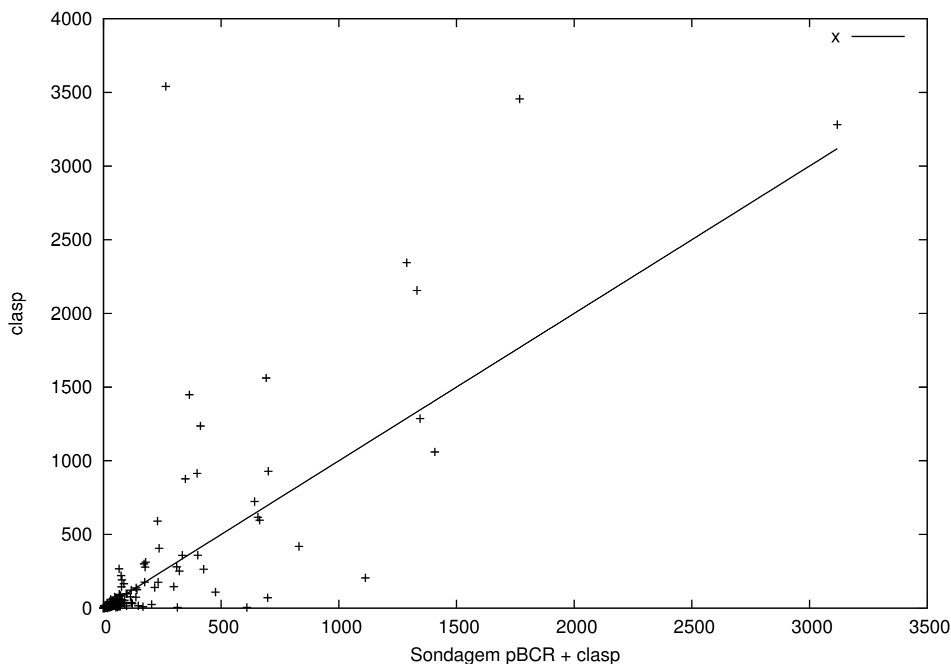

Figura 5.4: Comparação dos tempos de execução do CLASP nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência.

CLASP variaram. Primeiramente, o CLASP utilizando as fórmulas CNF escritas como pseudo-Boolean teve 19 fórmulas com o limite de tempo excedido, contra 17 quando se resolvidas em CNF. Dentre as sondadas, o CLASP em pseudo-Boolean ficou com 16 com limite de tempo, contra 13 do CLASP nas fórmulas sondadas em CNF. O CLASP em pseudo-Boolean resolveu 3 fórmulas que em suas versões sondadas não foram resolvidas, enquanto que resolveu 6 fórmulas sondadas que não foram resolvidas nas versões originais. O CLASP pseudo-Booleano resolveu 528 fórmulas mutuamente com o CLASP pseudo-Booleano sondado.

O CLASP com fórmulas pseudo-Booleanas conseguiu resolver 1 fórmula que sua versão em CNF não conseguiu dentro do limite de tempo, enquanto que a versão CNF resolveu 3 fórmulas que na versão pseudo-Booleana estouram o limite de tempo. Quando comparamos o CLASP em CNF com o CLASP pseudo-Booleano nas fórmulas sondadas, temos que esta última resolveu 5 que a versão CNF não resolveu, e a versão CNF resolveu 4 que a versão pseudo-Booleana não resolveu. Então, mutuamente, temos que o CLASP em

CNF com a versão pseudo-Booleana resolveram 530 fórmulas e a versão CNF comparada com a versão pseudo-Booleana sondada o número foi de 529 fórmulas. Comparando a versão CNF sondada com a pseudo-Booleana tivemos 530 mutuamente resolvidas e 534 mutuamente resolvidas quando comparamos a versão sondada em CNF com a pseudo-Booleana.

A figura 5.5 mostra a comparação do CLASP executado nas fórmulas CNF escritas em pseudo-Boolean com o CLASP executado nestas mesmas fórmulas após a sondagem do *pBCR*. O eixo x indexou os tempos utilizados pelo CLASP nas fórmulas CNF escritas em pseudo-Boolean sondadas, enquanto que no eixo y são sem sondagem. Quando os pontos estão acima da linha, temos que os tempos nas fórmulas sondadas são melhores.

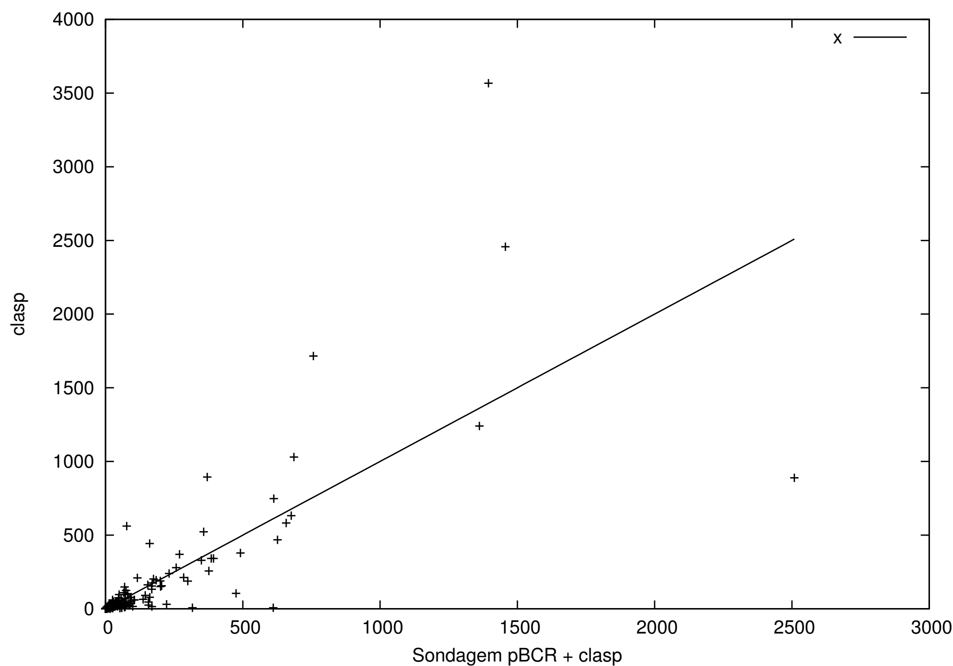

Figura 5.5: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência.

GLUCOSE

O resolvedor GLUCOSE conseguiu resolver todas as 549 fórmulas dentro do limite de tempo estipulado em ambas configurações, com e sem as fórmulas sondadas.

O tempo total gasto pelo GLUCOSE nas fórmulas originais foi de $34.588, 9$ segundos e

de 29.727, 8 segundos em cima das fórmulas sondadas, causando uma redução de 14, 05% no tempo de solução das fórmulas. Quando se considerou o tempo do *pBCR* executando a sondagem, o tempo total, i.e, tempo do GLUCOSE nas fórmulas mais o tempo do *pBCR* nas fórmulas tivemos 37.447, 1 segundos, tendo o *pBCR* levado 7.719, 35 segundos de processamento para a sondagem. Considerando o tempo do *pBCR* o GLUCOSE gerou um acréscimo de 8, 26% de tempo em relação a execução das fórmulas originais.

Nas 549 fórmulas, quando não se considerou o tempo da sondagem, as fórmulas pré-processadas obtiveram menor tempo de execução com o GLUCOSE em 388 delas, mas quando se considerou o tempo de sondagem o tempo utilizado pelo *pBCR* e GLUCOSE ganhou somente em 110 fórmulas enquanto que o GLUCOSE nas fórmulas originais ganhou nas 439. Na figura5.6 tivemos a comparação dos tempos de execução, ficando o tempo do GLUCOSE, somado ao tempo *pBCR* no eixo x e no eixo y o tempo do GLUCOSE nas fórmulas originais. Quando o ponto estiver acima da linha $f(x) = x$, significa que o GLUCOSE com o *pBCR* obteve um tempo inferior de processamento.

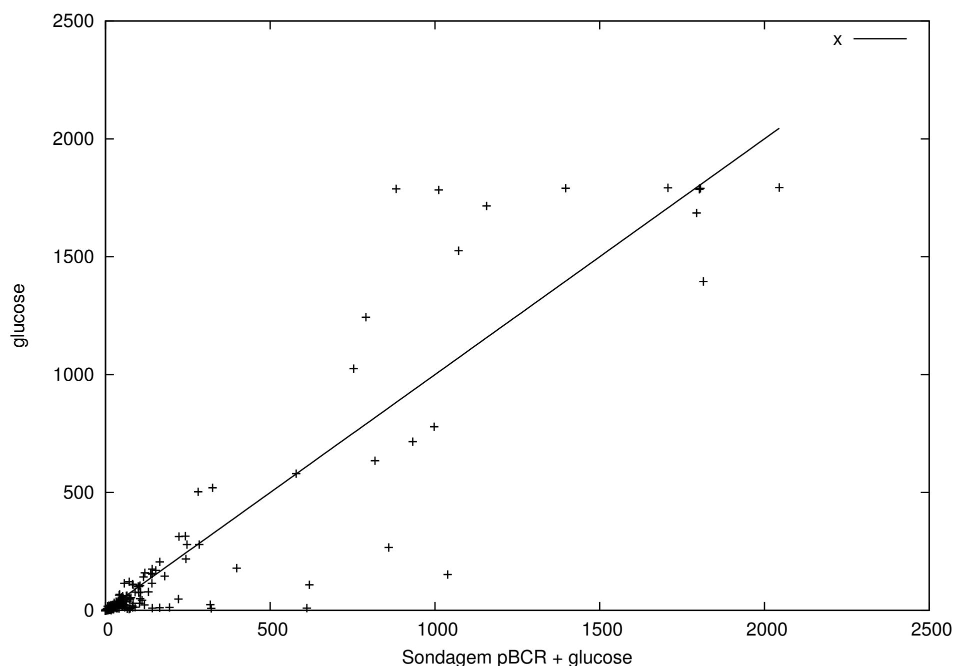

Figura 5.6: Comparação dos tempos de execução do GLUCOSE nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do GLUCOSE no eixo y contra o GLUCOSE com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência.

Quando consideramos os resolvedores individualmente na configuração original, i.e, diante das fórmulas originais, pudemos concluir que:

- o GLUCOSE foi o melhor resolvedor, pois ele resolveu todas as 549 fórmulas dentro do tempo limite;
- em segundo lugar o LINGELING, pois estourou o tempo em 10 fórmulas;e
- em terceiro lugar o CLASP, tendo 17 fórmulas não resolvidas dentro do limite de tempo;

Mas quando olhamos o tempo individual por fórmula por resolvedores, identificamos que cada resolvedor conseguiu suas próprias vitórias e derrotas neste contexto micro, ou seja, sem observar o conjunto completo das fórmulas.

O GLUCOSE quando comparado com os outros resolvedores, nas 549 fórmulas obteve o seguinte resultado: o GLUCOSE foi melhor em 329 quando comparado ao CLASP; foi melhor em 457 quando se comparado ao LINGELING. O CLASP comparado ao LINGELING foi melhor em 435 fórmulas.

Quando se comparou com as fórmulas sondadas o GLUCOSE foi melhor em 303 em relação ao CLASP e 441 contra o LINGELING. O CLASP foi melhor em 426 se comparado ao LINGELING.

Para melhor identificar os resolvedores em um âmbito geral, geramos a classificação média dos resolvedores em todas as configurações, em fórmulas originais e sondadas, e obtivemos a classificação descrita na tabela 5.3, e partir dela executamos o teste estatístico de Friedman e obtivemos o valor F de $31,83$, indicando que existe diferença estatística entre os métodos e com isso executamos o pós-teste de Nemenyi para gerar a diferença crítica entre eles, como representado na figura 5.7. Neste teste fica evidente que a realização da sondagem nas fórmulas impacta diretamente no tempo de execução do resolvedor, onde os tempos dos resolvedores nas fórmulas sondadas, em sua maioria, são menores que nas fórmulas originais.

Quando o tempo de pré-processamento do *pBCR* na sondagem foi considerado, os valores modificaram um pouco, como apresentado anteriormente, já que o tempo agregado

Resolvedor	Classificação Média
GLUCOSE+pbcr	2,131542
CLASP+pbcr	2,653855
GLUCOSE	2,835549
CLASP	3,394141
LINGELING+pbcr	4,337826
LINGELING	4,949454

Tabela 5.3: Classificação média dos resolvedores LINGELING, CLASP, GLUCOSE nas fórmulas originais e sondadas sem considerar o tempo de processamento do *pBCR*

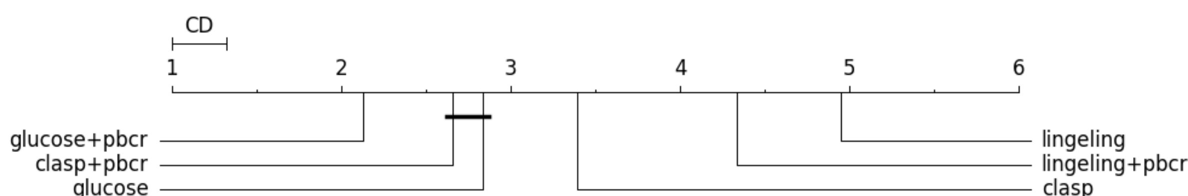

Figura 5.7: Comparação estatística do pós-teste de Nemenyi com os resolvedores LINGELING, CLASP e GLUCOSE em cima das fórmulas originais e pré-processadas pelo *pBCR* executando apenas a etapa 1, sondagem, sem considerar o tempo de processamento do *pBCR*

pelo *pBCR* nestas fórmulas afetou o tempo global de processamento. Para isso fizemos a classificação média dos resolvedores. Quando se contou o tempo do *pBCR*, conforme apresentado na tabela 5.4. O Friedman para essa classificação média teve o valor F de $80,97$, indicando a existência de diferença estatística entre os métodos e a figura 5.8 mostrando a diferença crítica gerada por Nemenyi. Nesta nova avaliação foi clara a diferença de quando não se considera o tempo da sondagem do *pBCR*. Porém é importante notar que com os resolvedores LINGELING e CLASP outras fórmulas foram resolvidas em suas versões sondadas que não puderam ser resolvidas no tempo limite estabelecido, representando, sim, uma vitória quando se compara a quantidade de problemas resolvidos dentro do tempo estabelecido.

5.1.2 Fórmulas Pseudo-Booleanas

Para os testes pseudo-Booleanos, executamos 766 fórmulas com os resolvedores CLASP e SCIP. Dentre estas fórmulas, 116 fórmulas foram exclusivamente de decisão, i.e, sem função objetivo, e por isso foram separadas das 650 fórmulas que possuem função objetivo.

Resolvedor	Classificação Média
GLUCOSE	1,979690
CLASP	2,603400
GLUCOSE+pbcr	3,210534
CLASP+pbcr	3,456315
LINGELING	4,331239
LINGELING+pbcr	4,903339

Tabela 5.4: Classificação média dos resolvedores LINGELING, CLASP, GLUCOSE nas fórmulas originais e sondadas considerando o tempo de processamento do *pBCR*

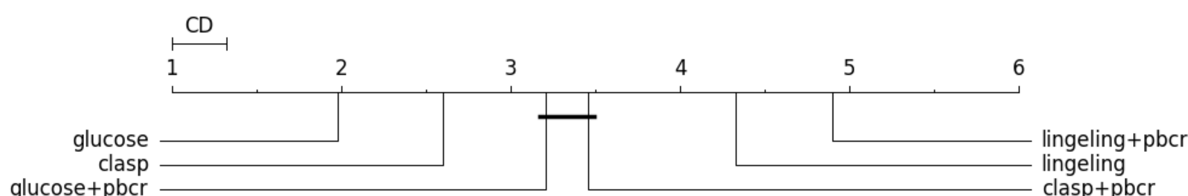

Figura 5.8: Comparação estatística do pós-teste de Nemenyi com os resolvedores LINGELING, CLASP e GLUCOSE em cima das fórmulas originais e pré-processadas pelo *pBCR* executando apenas a etapa 1, sondagem, considerando o tempo de processamento do *pBCR*

Desta forma o experimento foi dividido em duas etapas de decisão e otimização.

De todas as fórmulas, duas (2) foram identificadas como trivialmente insatisfatíveis pelo *pBCR*. O tempo de sondagem de ambas as fórmulas foi de 0 segundos, bem como o tempo de solução destas fórmulas pelo CLASP e SCIP. A partir deste ponto considerarmos apenas 114 fórmulas de decisão.

Decisão

Das fórmulas avaliadas o *pBCR* demorou $219,76$ segundos para executar a sondagem. Após isso, seguimos os passos feitos pelas fórmulas CNF e executamos os resolvedores CLASP e SCIP com as fórmulas originais e em suas versões sondadas.

CLASP

Dentre às 114 fórmulas, o CLASP não conseguiu decidir 10 fórmulas e dentre as fórmulas sondadas, também, 10 não foram decididas no tempo limite estipulado. Mas em duas fórmulas aconteceu uma inversão, em uma delas o CLASP conseguiu resolver em 2.925 segundos, enquanto que em cima da fórmula sondada o CLASP já não conseguiu decidí-la em 3.600 segundos. Em outra fórmula o CLASP não conseguiu decidir a fórmula

original dentro dos 3.600 segundos, mas a versão sondada foi resolvida em 495 segundos e o tempo de sondagem foi de $0,1$ segundos.

O tempo total de processamento do CLASP nas 103 fórmulas resolvidas mutuamente pelas duas configurações foi de $14.114,5$ segundos nas fórmulas originais e 14.635 segundos nas fórmulas já sondadas, gerando um acréscimo de $3,68\%$ no tempo execução. E $14.853,8$ considerando o tempo de processamento da sondagem, gerou um acréscimo de $5,23\%$ no tempo global de execução do CLASP com o tempo agregado da sondagem pelo *pBCR*.

A figura 5.9 representa o comparativo do CLASP na execução das fórmulas originais e sondadas, sempre considerando o tempo de processamento do *pBCR*, e sempre que o ponto estiver acima da linha significa que o tempo de processamento com o *pBCR* foi menor se comparado com a fórmula original. Dentre as 103 fórmulas resolvidas em ambas configurações o CLASP utilizou menos tempo de processamento em 70 fórmulas sondadas quando não se considerou o tempo de processamento do *pBCR*, mas este número reduz para 40 quando se considerou o tempo utilizado pela sondagem.

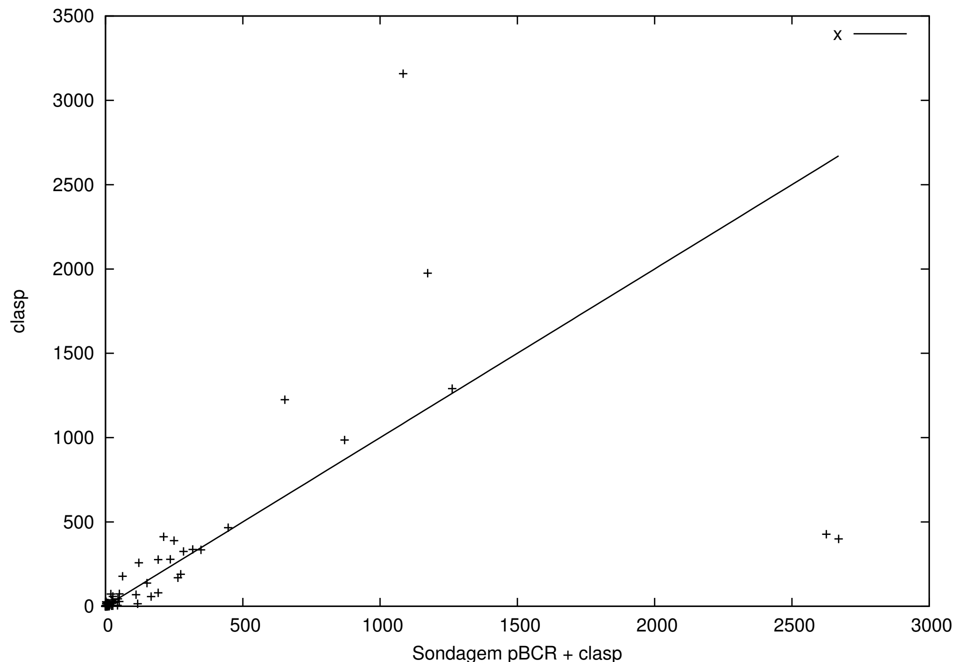

Figura 5.9: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência.

SCIP

Dentre às 114 fórmulas, o SCIP não conseguiu decidir 82 fórmulas, e dentre as fórmulas sondadas, 83 não foram decididas no tempo limite estipulado.

O tempo total de processamento do SCIP foi de 300.384 segundos nas fórmulas originais e de 302.672 nas fórmulas sondadas, gerando um acréscimo no tempo de execução de $0,76\%$. Tendo adicionado o tempo de processamento do *pBCR* o tempo total passou a ser 302.819 e o acréscimo passou ser de $0,81\%$ em relação à execução nas fórmulas originais.

Otimização

O conjunto de fórmulas que possuem uma função objetivo a ser minimizada foi de 650 fórmulas. Estas fórmulas foram extremamente fáceis de serem decididas levando em média 5 segundos de processamento, porém, o tempo de otimização foi muito maior, ficando poucas fórmulas com a solução ótima encontrada no tempo limite definido, como veremos a seguir.

O *pBCR* utilizou $1.145,76$ segundos de processamento para executar a sondagem em todas as 650 fórmulas, ficando uma média de $1,7$ segundos por fórmula, sendo que somente 15 fórmulas superaram 3 segundos de processamento e somente 5 utilizaram 12 segundos de processamento.

CLASP

O CLASP encontrou a solução ótima em 79 fórmulas dentro do tempo limite, levando ao todo 25.313 segundos de processamento para essas fórmulas.

Quando se utilizou as fórmulas sondadas, 78 fórmulas o ótimo foi encontrado dentro do tempo limite, e o tempo total foi de 28.890 segundos, ficando $14,13\%$ acima do tempo em relação as fórmulas originais.

Das fórmulas originais, que foram encontradas a solução ótima, 15 não foram provadas o ótimo na versão sondada. Já das fórmulas sondadas, 14 não foram provadas ótimas na fórmula original. As 15 fórmulas provadas ótimas em suas versões sem modificação utilizaram $8.157,22$ segundos ao todo, e as 14 que foram provadas ótimas apenas na versão modificada pela sondagem utilizaram $11.977,9$ segundos de processamento e $12.001,7$ segundos quando se considerou o tempo utilizado pela sondagem.

Com essas variações restaram 64 fórmulas que são provadas ótimas em ambas configurações dentro do limite de tempo estabelecido. Destas fórmulas, 40 foram provadas ótimas em menos tempo em sua varição sondada, e este número reduzido para 24 quando se considerou o tempo de sondagem utilizado pelo *pBCR*. A figura 5.10 mostra a comparação dos tempos destas 64 fórmulas tendo o eixo x indexado o tempo de execução das fórmulas sondadas pelo CLASP e considerando o tempo de execução do *pBCR*, e sempre que o ponto estiver acima da linha significa que o tempo da fórmula sondada foi inferior ao da fórmula original.

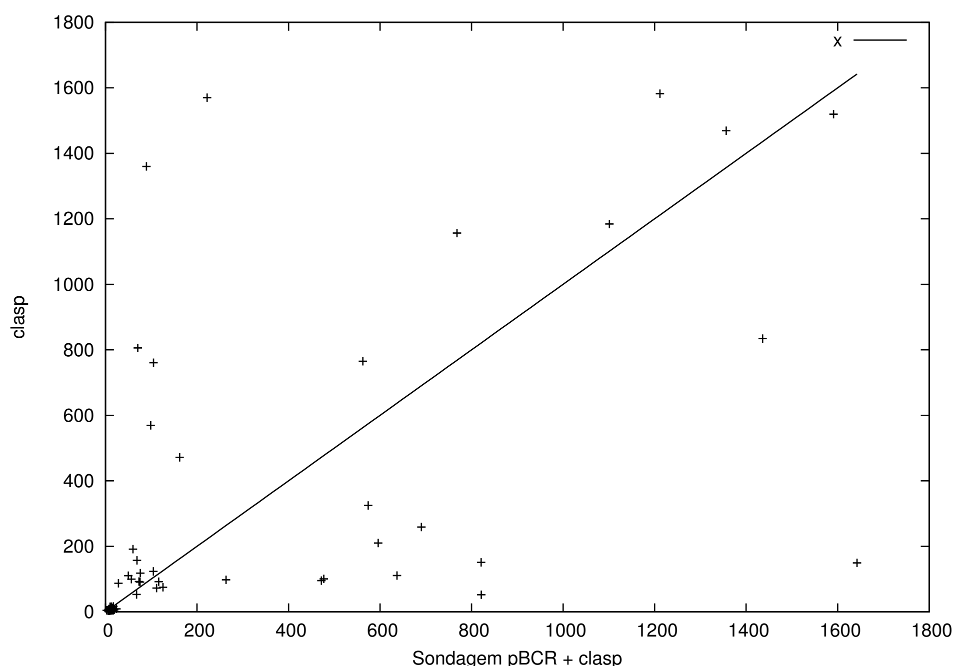

Figura 5.10: Comparação dos tempos de execução do CLASP nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência.

SCIP

Utilizando o resolvedor SCIP, temos 239 fórmulas com o ótimo provado dentro do tempo estipulado, e com as fórmulas sondadas passaram a 254 fórmulas com o ótimo provado.

Das fórmulas originais que foram provadas ótimas, 7 não foram provadas ótimas na versão sondada. E 22 fórmulas que o *pBCR* executou a sondagem o ótimo foi encontrado

mas não foram nas fórmulas originais.

O SCIP usou 141.976 segundos de processamento para identificar o ótimo nas 239 fórmulas originais e usou 130.665 segundos nas 254 fórmulas sondadas. Dentre as fórmulas que foram provados o ótimo em ambas configurações, fórmula original e fórmula sondada, obtemos 133.059 segundos nas fórmulas originais e 105.312 segundos nas fórmulas sondadas, o *pBCR* utilizou 427 segundos para efetuar a sondagem nestas fórmulas, sendo este conjunto mútuo formado por 232 fórmulas. As fórmulas sondadas necessitaram de $20,8\%$ menos tempo para provar o ótimo, quando não se levou em consideração o tempo de processamento da sondagem do *pBCR*, e mesmo considerando este tempo a redução foi de $20,5\%$.

Na figura 5.11 cada ponto representa uma fórmula, e nesta figura estão representados as 232 fórmulas que são provadas ótimas por ambos, SCIP nas fórmulas originais e o SCIP nas fórmulas sondadas pelo *pBCR*. O tempo de otimização feito pelo SCIP nas fórmulas sondadas é indexado pelo eixo x, enquanto que nas fórmulas originais é indexado no eixo y. A linha $f(x) = x$ foi traçada como referência para auxiliar na identificação das fórmulas que são melhores quando sondadas e originais. Todos os pontos acima da linha representam que a fórmula sondada teve tempo inferior ao das fórmulas originais.

5.2 Teste de Aprendizado

No *pBCR* foi implementado o método de aprendizado simplificado, conforme explicado na seção 4.2 e mais especificamente o algoritmo 6 descrito em detalhes na seção.

Para o experimento do aprendizado o objetivo é avaliar o impacto no tempo de decisão e otimização quando adicionamos as restrições aprendidas na fórmula. Como o aprendizado é dependente da etapa de sondagem, as fórmulas que terão as restrições adicionadas serão as que foram sondadas, i.e, sem desligar o efeito da sondagem nas fórmulas. A figura 5.12 apresenta em que ponto este teste está presente no âmbito global de resolução de uma fórmula.

Dentre as fórmulas testadas, não foram consideradas no processo todas as fórmulas que foram identificadas como trivialmente insatisfatível. Ficaram 549 fórmulas do benchmark

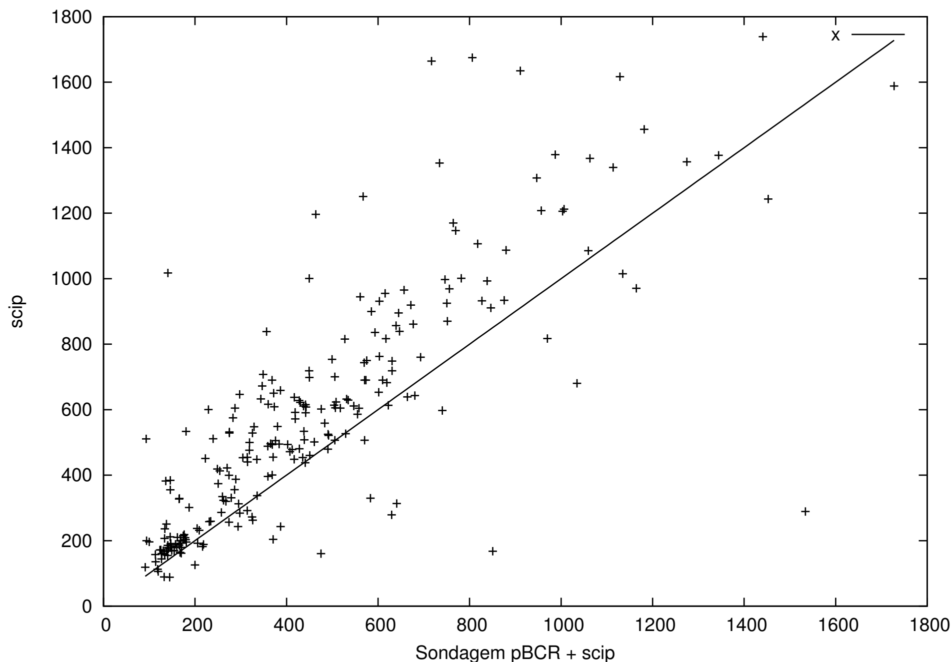

Figura 5.11: Comparação dos tempos de execução do SCIP nas fórmulas originais e sondadas pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP no eixo y contra o SCIP com as fórmulas sondadas no eixo x. A linha $f(x) = x$ é colocada como referência.

CNF, 114 do benchmark de decisão pseudo-Booleano e 650 fórmulas do benchmark de otimização pseudo-Booleana.

5.2.1 Fórmulas CNF

Neste experimento as fórmulas CNF ficaram, necessariamente, uma fórmula pseudo-Booleana devido à incorporação das restrições aprendidas que foram pseudo-Booleanas, como explicado anteriormente. Por este motivo os resolvedores em CNF não puderam ser utilizados, a menos que alguma transformação fosse feita na fórmula, mas este não é o foco desta tese. E também as fórmulas passadas ao CLASP são sempre escritas em pseudo-Boolean, mesmo as sem modificação feita pelo *pBCR*.

O tempo que o *pBCR* utilizou para pré-processar as 549 fórmulas foi de $8.903,45$ segundos, um acréscimo de $15,33\%$ quando se comparado com o *pBCR* rodando apenas a etapa da sondagem.

Para as 549 fórmulas oriundas do benchmark CNF o CLASP conseguiu resolver 530, ficando 19 sem solução dentro do tempo limite de tempo. Quando se contou nas fórmulas

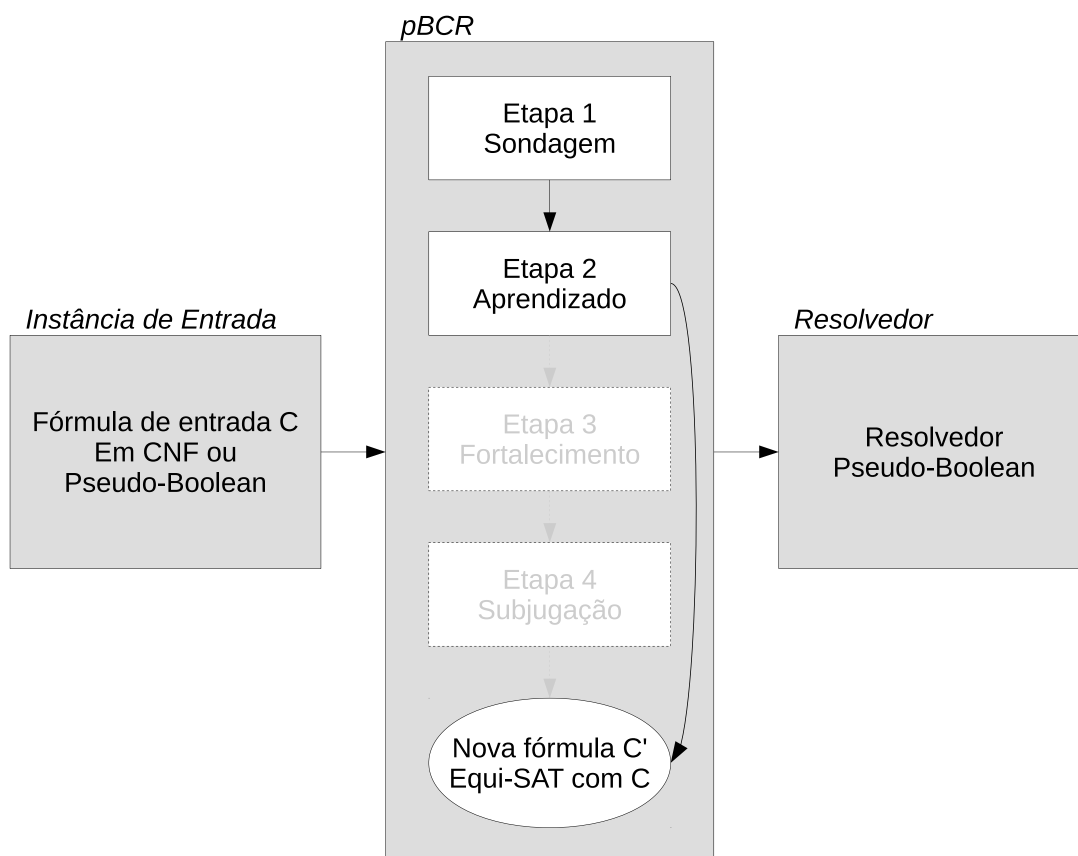

Figura 5.12: Diagrama do processo de aplicação do pré-processador *pBCR* no experimento da Aprendizado, etapa 2

com o aprendizado injetado na fórmula tivemos que o CLASP conseguiu resolver 534. 15 não foram resolvidas. Considerando as duas configurações o CLASP resolveu 529 fórmulas mutuamente entre a fórmula original e com as restrições aprendidas injetadas na fórmula.

O CLASP nas fórmulas originais conseguiu resolver uma fórmula, que a versão com o aprendizado não conseguiu resolver dentro do tempo limite. Já com o aprendizado o CLASP conseguiu resolver 5 fórmulas a mais, se comparado às fórmulas originais. Se comparado com as 529 fórmulas que ambas configurações resolvem, 333 fórmulas foram resolvidas mais rapidamente com a versão, com as restrições adicionais quando não se considerou o tempo do *pBCR*. Quando este tempo foi agregado 105 fórmulas destas 529 foram resolvidas mais rapidamente que as fórmulas originais.

O tempo total do CLASP nas 529 fórmulas decididas por ambas configurações foi de $29.971,9$ segundos nas fórmulas originais, de $35.976,4$ segundos nas fórmulas com as

restrições adicionais, um acréscimo de 20,03%, e 42.589 segundos quando se considerou o tempo de processamento do *pBCR*, causando um acréscimo ainda maior, de 42,09% no tempo global.

A figura 5.13 mostra o desempenho do CLASP executado nas fórmulas originais escritas em pseudo-Boolean, indexadas no eixo y, contra as fórmulas com as restrições aprendidas adicionadas às fórmulas, indexadas no eixo x. Sempre que o ponto estiver acima da linha, representa um menor tempo de execução com o pré-processamento do *pBCR*. Neste gráfico o tempo utilizado pelo *pBCR* está agregado na conta total.

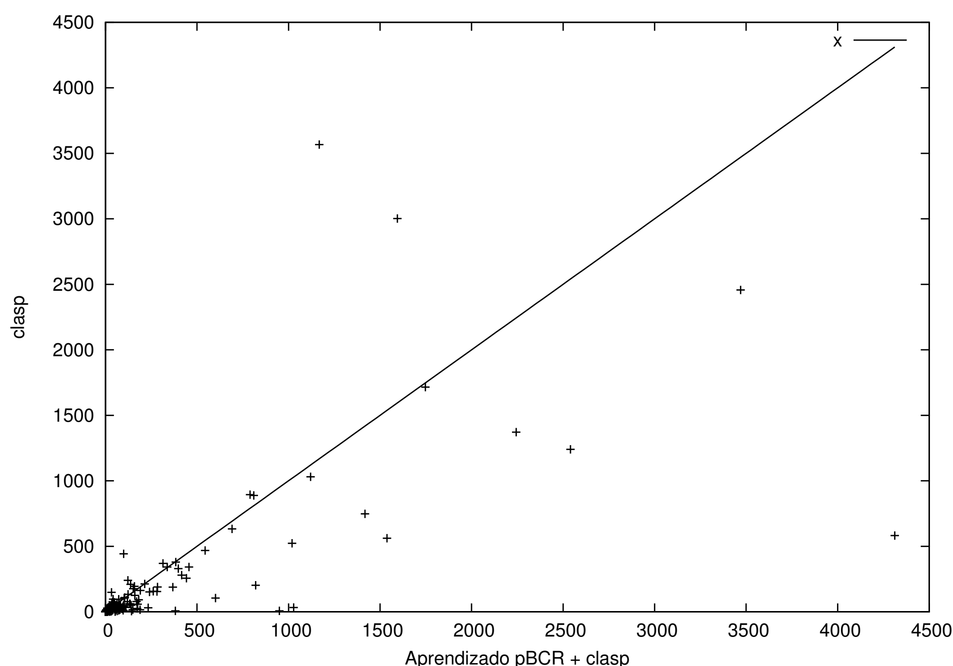

Figura 5.13: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, do benchmark CNF, originais e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP no eixo y contra o CLASP com as fórmulas com as restrições aprendidas no eixo x. A linha $f(x) = x$ é colocada como referência.

Comparando o aprendizado com a sondagem temos que 531 fórmulas foram mutuamente resolvidas, sendo que 2 fórmulas com aprendizado não foram resolvidas mas foram quando são sondadas.E 3 fórmulas foram resolvidas com as restrições aprendidas que não foram resolvidas com apenas a sondagem realizada. E entre as fórmulas mutuamente resolvidas com essas configurações o CLASP em cima das fórmulas com o aprendizado foram mais rápidas em 155 delas.

A figura 5.14 compara o desempenho do CLASP em cima das fórmulas sondadas, indexado no eixo y, e o CLASP com as fórmulas com as restrições adicionadas, no eixo x. Sempre que o ponto estiver acima da linha representa que o tempo de execução do CLASP foi menor quando executado com as fórmulas com as restrições adicionadas. Neste gráfico o tempo de processamento do *pBCR* foi desconsiderado, já que o objetivo foi apenas mostrar a qualidade das fórmulas com a ativação de mais um passo do *pBCR*.

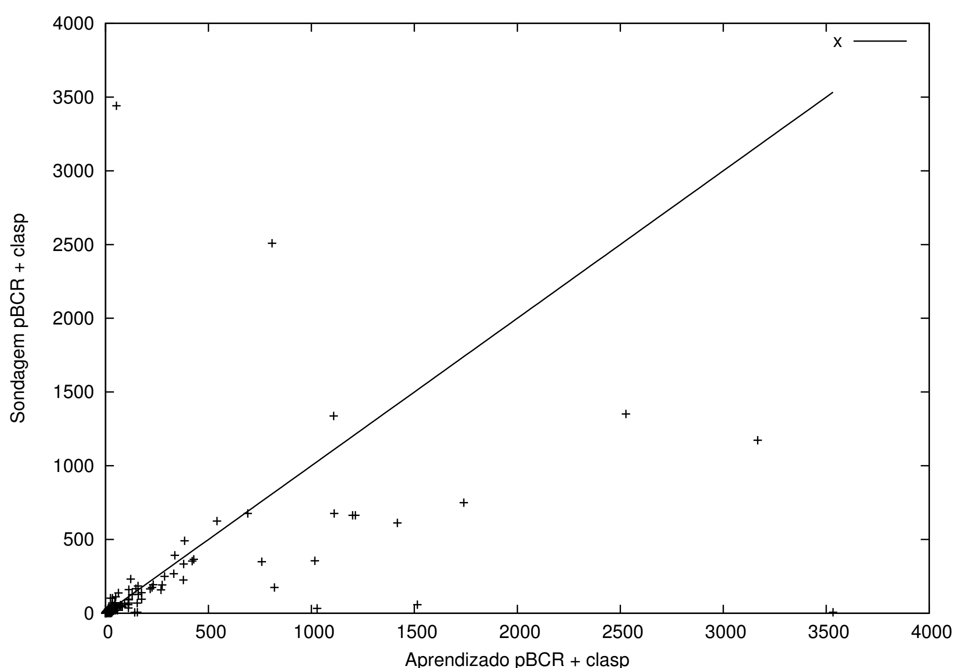

Figura 5.14: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, do benchmark CNF, sondadas e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas sondadas no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência.

5.2.2 Fórmulas pseudo-Booleanas

Para os experimentos com as fórmulas pseudo-Booleanas, usaremos as 114 fórmulas de decisão que não foram identificadas como trivialmente insatisfatíveis e as 650 fórmulas de otimização.

Decisão

O tempo que o *pBCR* utilizou para pré-processar as 114 fórmulas de decisão foi de 460, 21 segundos, agregando 240, 45 segundos em relação à etapa da sondagem.

O desempenho do CLASP foi de 102 fórmulas resolvidas em 17.489 segundos. Com as fórmulas aprendidas o CLASP conseguiu resolver 1 fórmula que o CLASP nas fórmulas originais não conseguiu resolver. Porém, 3 fórmulas que foram resolvidas com o CLASP nas fórmulas originais não foram resolvidas nas fórmulas aprendidas, foram 101 fórmulas resolvidas mutuamente entre estas duas configurações. O CLASP nestas 101 fórmulas originais utilizou 11.711, 8 segundos de processamento, nas fórmulas aprendidas 16.497, 9 segundos e 16.736, 5 segundos quando se considerou o tempo de processamento do *pBCR*, ficando um acréscimo de 42% no tempo total.

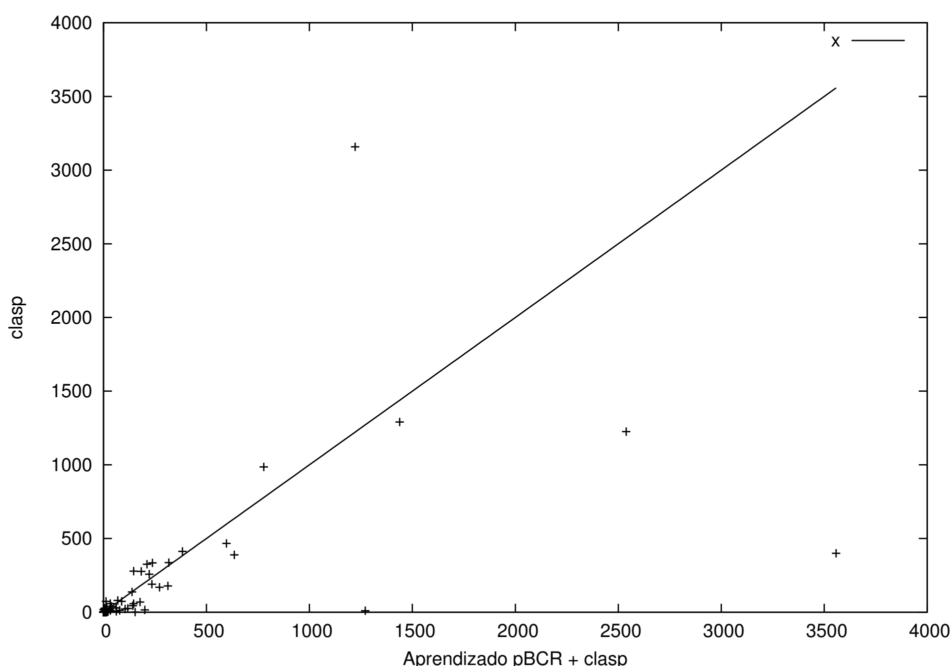

Figura 5.15: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, originais e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência.

Comparando com a execução do CLASP com as fórmulas com as restrições aprendidas e sondadas, tivemos 102 fórmulas resolvidas mutuamente entre as configurações, Foram 2 as fórmulas que estouram o limite de tempo com a versão aprendida e que foram

resolvidas quando se utilizadou a sondagem. Não houve fórmula resolvida na versão aprendida que não tenha sido resolvida na versão sondada. Destas 102 fórmulas o CLASP com as fórmulas com as restrições aprendidas o tempo de processamento foi de 17.489, contra 11.331.8 segundos quando executadas nas fórmulas sondadas, então adicionando o aprendizado houve um acréscimo de 54% no tempo total.

A figura 5.16 apresentou os tempos de execução do CLASP nas fórmulas com restrições aprendidas, indexadas no eixo x, contra as fórmulas sondadas, indexadas no eixo y, sem considerar o tempo utilizado pelo *pBCR*. Sempre que o ponto estiver acima da linhas o tempo utilizado pelo CLASP para decidir as fórmulas com restrições aprendidas foi menor que das sondadas.

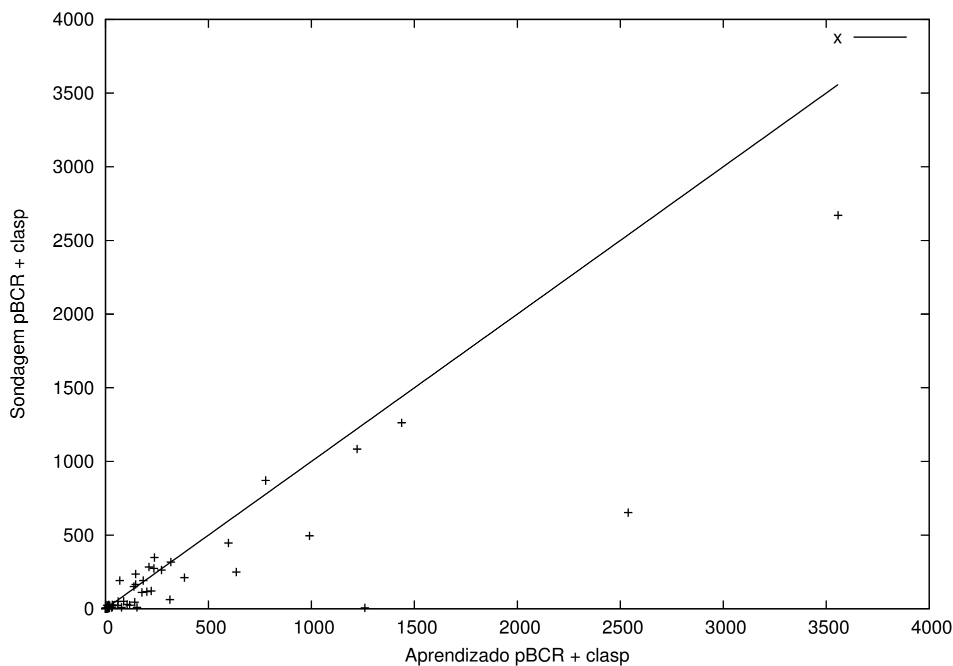

Figura 5.16: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, sondadas e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas sondadas no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência.

Otimização

Para as 650 fórmulas de otimização o *pBCR* utilizou $1.154,72$ segundos de processamento, um acréscimo de apenas $8,96$ segundos em relação à sondagem.

CLASP

O resolvedor CLASP encontrou solução ótima para 76 fórmulas com aprendizado, contra 78 quando se utilizou as fórmulas sondadas e 79 quando o CLASP executou nas fórmulas sem modificações feitas pelo *pBCR*.

Houve 64 fórmulas resolvidas mutuamente entre o CLASP nas fórmulas originais e nas fórmulas com as restrições aprendidas. O CLASP nas fórmulas com as restrições aprendidas resolveu 12 fórmulas que não foram resolvidas com as fórmulas originais e 15 foram resolvidas nas originai,s e não o foram nas com o aprendizado.

A figura 5.17 apresenta a comparação entre o CLASP nas fórmulas originais e com aprendizado. As fórmulas originais estão no eixo y e as com aprendizado no eixo x. Quando o ponto fica acima da linha significa que o tempo utilizado pelo CLASP nas fórmulas com as restrições aprendidas é menor.

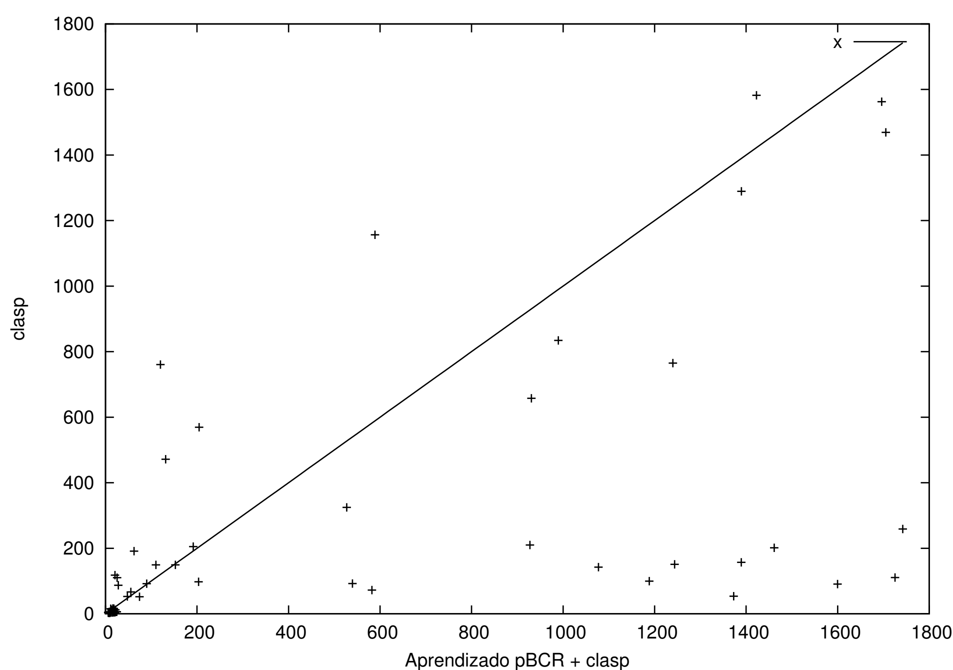

Figura 5.17: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, originais e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência.

Quando comparamos o CLASP executando nas fórmulas com sondagem com as restrições aprendidas, ocorreu que 61 fórmulas foram resolvidas mutuamente dentro do tempo

limite de tempo. Com as fórmulas aprendidas o CLASP conseguiu resolver 15 que não foram resolvidas com as fórmulas sondadas, enquanto que 17 foram resolvidas com aprendizado mas foram, quando apenas sondadas.

A figura 5.18 apresenta a comparação de tempo entre o CLASP nas fórmulas sondadas, no eixo y, e com as restrições aprendidas, no eixo x, e sempre que o ponto ficar acima da linha o tempo utilizado pleo CLASP nas fórmulas com aprendizado foi menor. O tempo do $pBCR$ não foi considerado neste gráfico.

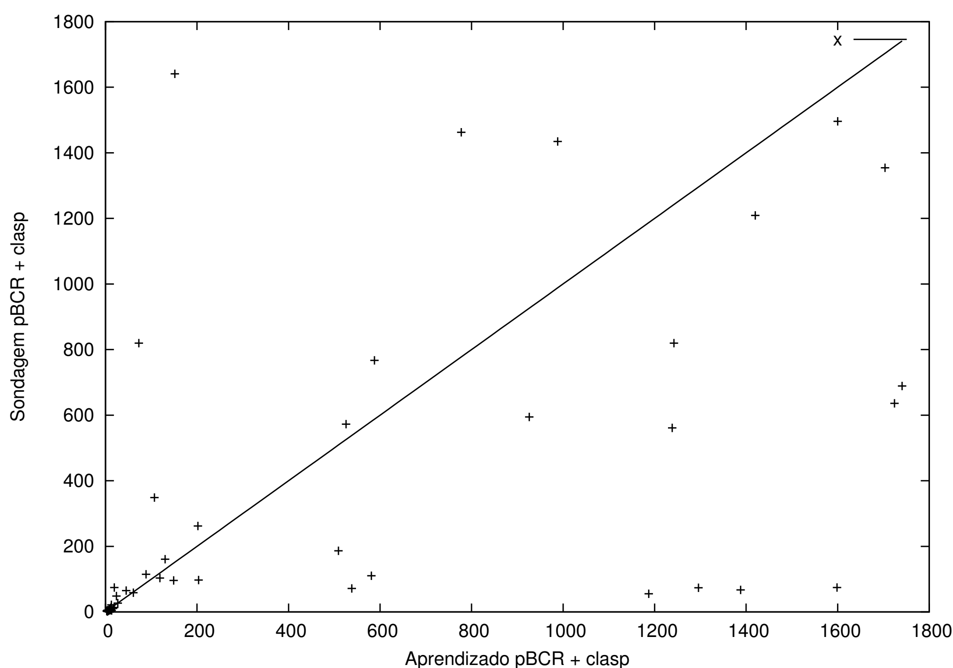

Figura 5.18: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas, sondadas e com aprendizado pelo $pBCR$. O gráfico mostra o tempo em segundos do CLASP com as fórmulas sondadas no eixo y contra o CLASP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência.

Comparando as três configurações, o CLASP com as fórmulas aprendidas resolveu 6 fórmulas que não foram resolvidas nem com as fórmulas originais, e nem com as fórmulas sondadas.

SCIP

O resolvedor SCIP encontrou solução ótima para 212 fórmulas quando executado nas fórmulas com aprendizado, contra 254 fórmulas dentre as sondadas e 239 quando se considerou as fórmulas sem modificação.

Dentre as 212 fórmulas resolvidas pelo SCIP com aprendizado, tivemos 202 resolvidas mutuamente com o SCIP nas fórmulas originais. O SCIP prova o ótimo em 37 fórmulas que com o aprendizado não conseguiu provar no tempo limite, e 10 fórmulas foram provadas ótimas com aprendizado, que não foram provadas com as fórmulas originais. O SCIP utilizou 106.303 segundos de processamento para resolver as 202 fórmulas e 155.752 segundos nas fórmulas com aprendizado, um acréscimo de $46,6\%$ no tempo global, quando se considerou o tempo utilizado pelo *pBCR* o tempo utilizado foi 156.135 segundos, o acréscimo foi de $46,8\%$, quando comparado com as fórmulas originais.

A figura 5.19 apresenta o desempenho do SCIP nas 202 fórmulas resolvidas mutuamente entre as configurações, nas fórmulas originais e com as restrições aprendidas. Sempre que o ponto estiver acima linha, temos que o tempo utilizado pelo SCIP nas fórmulas com aprendizado foi menor que nas fórmulas originais.

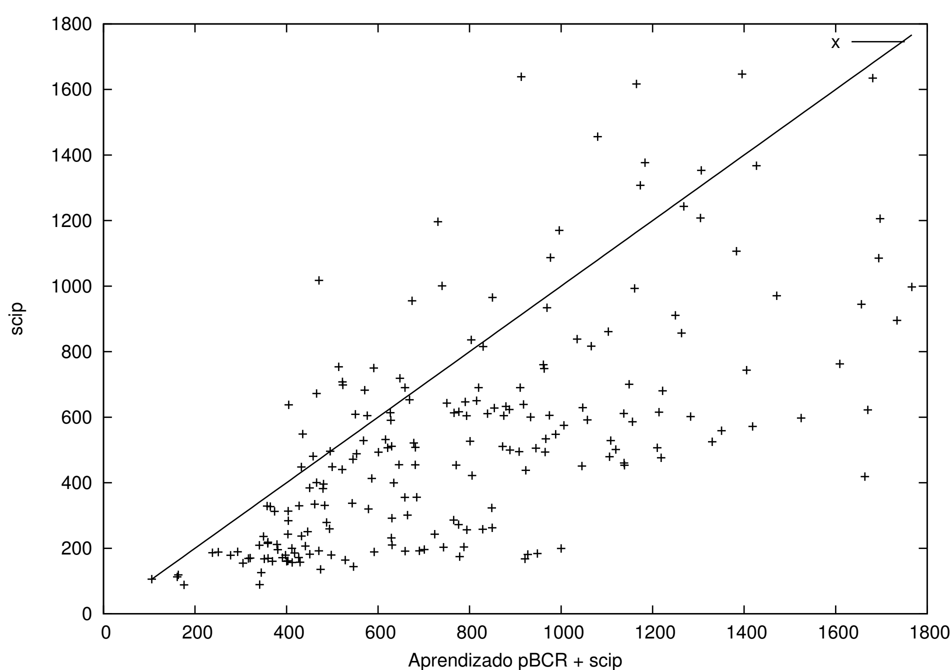

Figura 5.19: Comparação dos tempos de execução do SCIP nas fórmulas pseudo-Booleanas, originais e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP com as fórmulas originais no eixo y contra o SCIP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência.

Quando comparamos o desempenho do SCIP nas fórmulas com aprendizado e com as fórmulas sondadas, observamos que 206 fórmulas foram resolvidas mutuamente entre as

configurações, e 6 fórmulas com as restrições aprendidas foram provadas ótimas, enquanto que não foram quando apenas sondadas, e 48 foram provadas ótimas apenas sondadas e não o foram com as restrições aprendidas.

A figura 5.20 apresenta a comparação do resolvedor SCIP executando com as fórmulas sondadas, indexada no eixo y, e com as restrições aprendidas, indexada no eixo x. O tempo do *pBCR* não foi considerado neste gráfico.

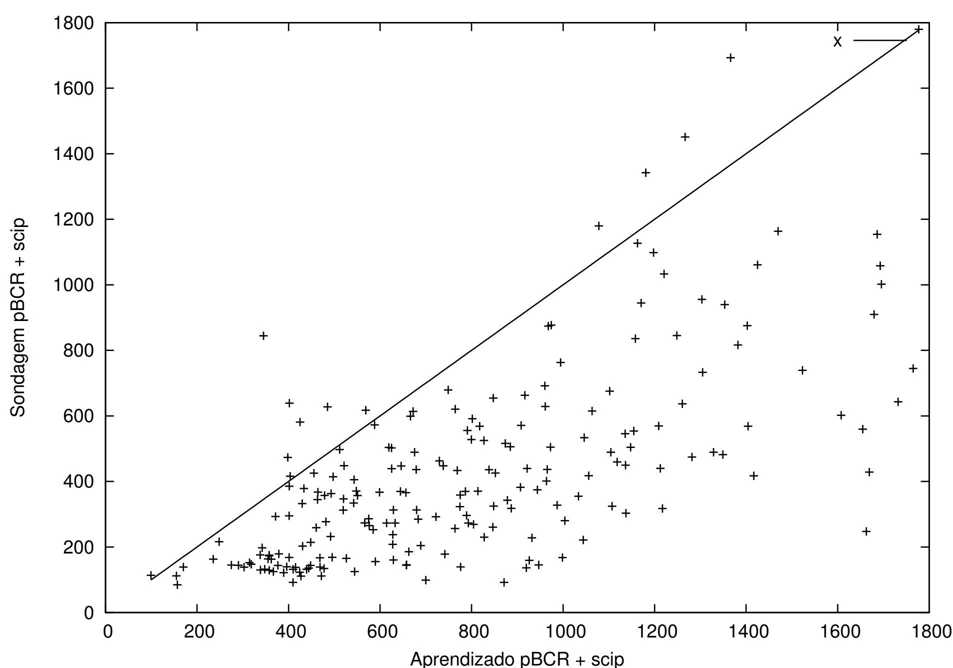

Figura 5.20: Comparação dos tempos de execução do SCIP nas fórmulas pseudo-Booleanas, sondadas e com aprendizado pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP com as fórmulas sondadas no eixo y contra o SCIP com as fórmulas com as restrições adicionais no eixo x. A linha $f(x) = x$ é colocada como referência.

Quando os tempos das três configurações, nas fórmulas originais, sondadas e com aprendizado foram cruzadas, identificamos que 4 fórmulas foram provadas ótimas apenas quando possuiam as restrições de aprendizado inseridas na fórmula.

5.3 Teste do Fortalecimento

No *pBCR* o fortalecimento também foi implementado e foi o foco deste experimento. Como esta etapa também depende da sondagem, a etapa 1 permaneceu ligada. A etapa 4, que diz respeito à subjugação, também foi ligada, uma vez que ela é um fator impor-

tante do processo de fortalecimento, onde se remove das fórmulas as restrições que são desnecessárias. A figura 5.21 é um diagrama que mostra as etapas ligadas e desligadas deste experimento.

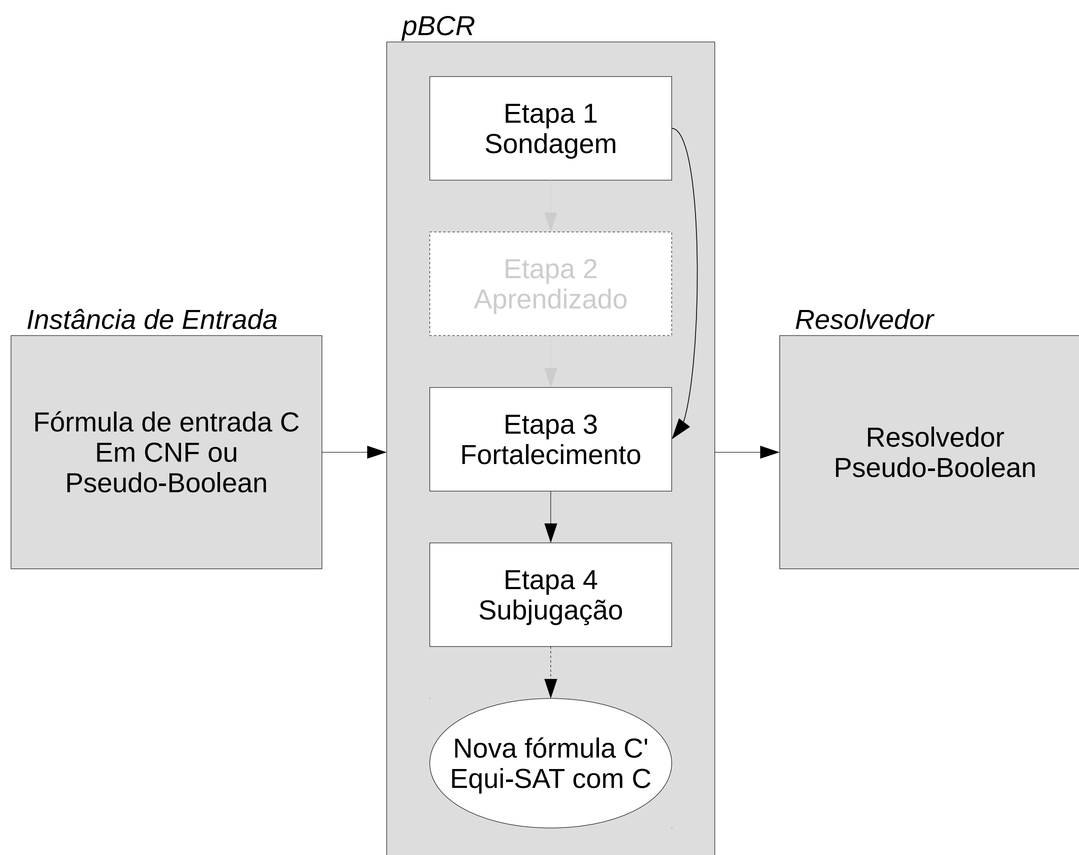

Figura 5.21: Diagrama do processo de aplicação do pré-processador *pBCR* no experimento do Fortalecimento, etapa 3

Para efetuar o experimento do fortalecimento limitamos o crescimento de literais nas restrições, divididos em três configurações: 2, 4 e 8. Este limite se dá por diversos fatores, primeiro é o limite de memória pois enquanto a fórmula está crescendo nenhuma restrição é cortada da fórmula e o consumo de memória cresce; também o tempo de processamento para efetuar o fortalecimento completo na restrição, já que quanto mais crescem as restrições mais tempo se utiliza para identificar a restrição supersatisfeita e o tempo fica mais perceptível nas fórmulas com muitas restrições.

As variações de crescimento limitado por restrição foram analisadas em paralelo nas seções a seguir.

5.3.1 Fórmulas CNF

Assim como o experimento do aprendizado, este experimento do fortalecimento também gerou fórmulas exclusivamente pseudo-Booleanas, tirando a possibilidade de se utilizar resolvedores CNF para resolver estas fórmulas. E por isso usamos o CLASP para este experimento, e sempre que comparado com as fórmulas originais estaremos considerando o CLASP executado com as fórmulas originais escritas no formato pseudo-Booleano, que tem um desempenho diferente do CLASP nas fórmulas escritas em CNF.

Dentre as 549 fórmulas em CNF que não foram identificadas como trivialmente insatisfatíveis na etapa de sondagem, 12 foram identificadas como trivialmente insatisfatíveis na etapa de fortalecimento. Desta forma restaram 537 fórmulas que foram pré-processadas pelo *pBCR*.

Para efetuar o fortalecimento com crescimento 2 foi utilizado 117.504 segundos de processamento pelo *pBCR* nestas 537 fórmulas. O crescimento 4 utilizou 161.866 segundos e o crescimento 8 foi de 246.321 segundos.

Dentre as 537 fórmulas, o fortalecimento de crescimento 2 não resolveu 47 fórmulas no tempo limite estipulado. De crescimento 4 tivemos 50 fórmulas e crescimento 8 com 55 fórmulas com limite de tempo excedido. O CLASP nas fórmulas originais não resolveu 16 fórmulas.

Comparando o CLASP nas fórmulas originais com o CLASP nas fórmulas fortalecidas com crescimento 2, identificmos que 488 fórmulas foram resolvidas mutuamente entre as configurações, 31 foram resolvidas com as fórmulas originais e não o foram com elas fortalecidas. E 2 fórmulas Foram resolvidas com elas fortalecidas e não o foram nas originais. O CLASP utilizou menos tempo de processamento em 267 fórmulas quando elas estavam fortalecidas, e quando se considerou o tempo utilizado pelo *pBCR*, apenas 53 fórmulas utilizavam menos tempo de processamento. A figura 5.22 mostra a comparação de tempo entre estas duas configurações, ficando no eixo x a configuração com as fórmulas fortalecidas.

Agora comparando o CLASP nas fórmulas originais com as fórmulas com fortalecimento de crescimento 4 tivemos 486 fórmulas resolvidas mutuamente entre as confi-

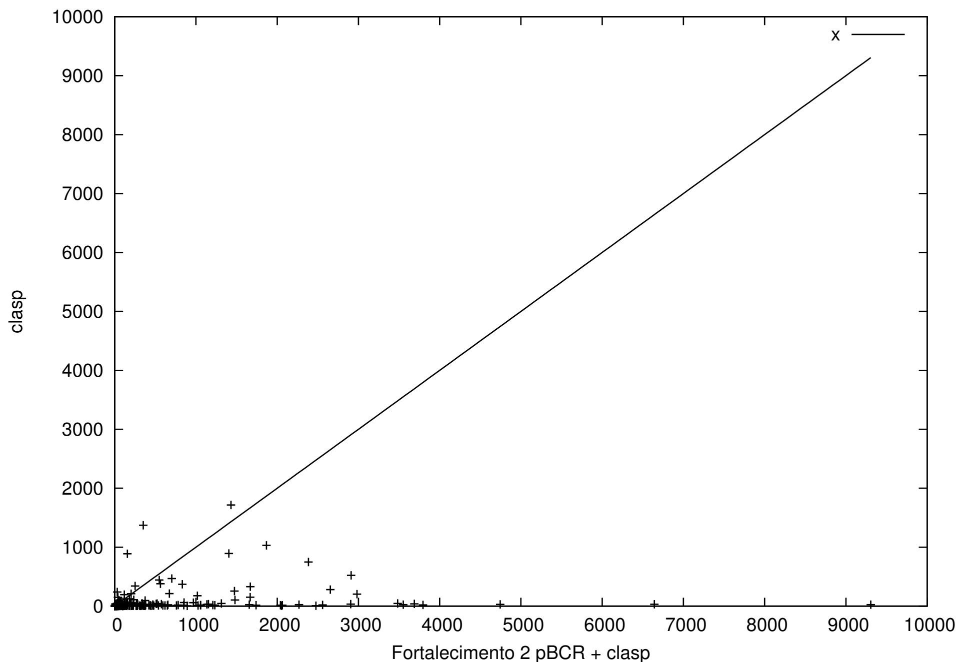

Figura 5.22: Comparação dos tempos de execução do CLASP nas fórmulas CNF originais e com fortalecimento de crescimento 2 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

gurações. São 33 fórmulas resolvidas pelo CLASP nas fórmulas originais que não foram resolvidas com o fortalecimento, e apenas 1 que foi resolvida com fortalecimento que não era com a fórmula original. Das fórmulas resolvidas em ambas configurações o CLASP utilizou menos tempo de processamento em 252 fórmulas fortalecidas, este número foi reduzido a 42 quando o tempo de processamento do *pBCR* foi considerado. A figura 5.23 apresentou o comparativo entre as configurações, sendo o eixo x indexando o tempo de processamento do CLASP nas fórmulas fortalecidas.

Quando comparamos o CLASP com o CLASP nas fórmulas fortalecidas de crescimento 8 notamos que 480 fórmulas foram resolvidas mutuamente. Foram 39 fórmulas resolvidas pelo CLASP nas fórmulas originais que não foram resolvidas com fortalecimento 8, e 2 fórmulas que foram resolvidas com o fortalecimento e não foram na original. Das fórmulas mutuamente resolvidas, o CLASP utilizou menos tempo de processamento em 217 e em 31 quando se considerou o tempo de processamento do *pBCR*. A figura 5.24 apresenta o comparativo entre as configurações, sendo o eixo x indexando o tempo de processamento

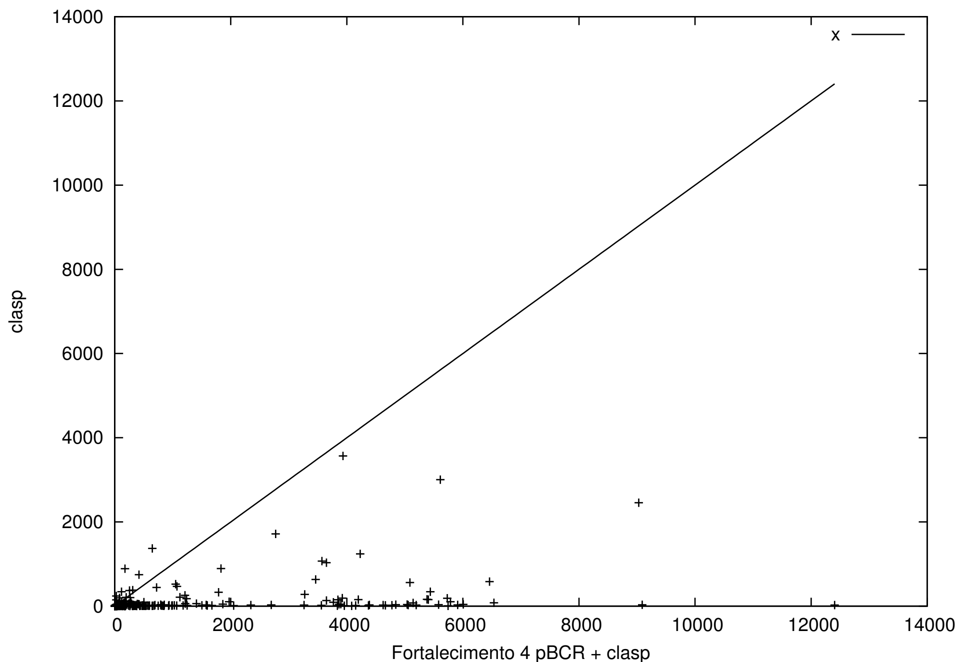

Figura 5.23: Comparação dos tempos de execução do CLASP nas fórmulas CNF originais e com fortalecimento de crescimento 4 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

do CLASP nas fórmulas fortalecidas.

5.3.2 Fórmulas pseudo-Boolean

Para as fórmulas pseudo-Booleanas utilizamos o mesmo sistema de avaliação que nas fórmulas CNF, com variações de 2, 4 e 8 para o máximo crescimento das restrições. Nas fórmulas de otimização temos apenas as variações de 2 e 4, pois não ocorre crescimento maior que 4 nas restrições com a configuração de sondagem utilizada.

Decisão

Para as 114 fórmulas de decisão o *pBCR* utilizou $479,80$, $497,42$ e $500,46$ segundos para, respectivamente, efetuar o fortalecimento nas configurações de 2, 4 e 8.

O CLASP resolveu 105 fórmulas das 114 com o fortalecimento 2, tendo 103 em comum com o CLASP nas fórmulas originais. Apenas 1 fórmula foi resolvida pelo CLASP na fórmula original que não foi resolvida com o fortalecimento e, 2 fórmulas não foram resolvidas com as fórmulas originais, mas o foram com as fórmulas fortalecidas. Dentre as

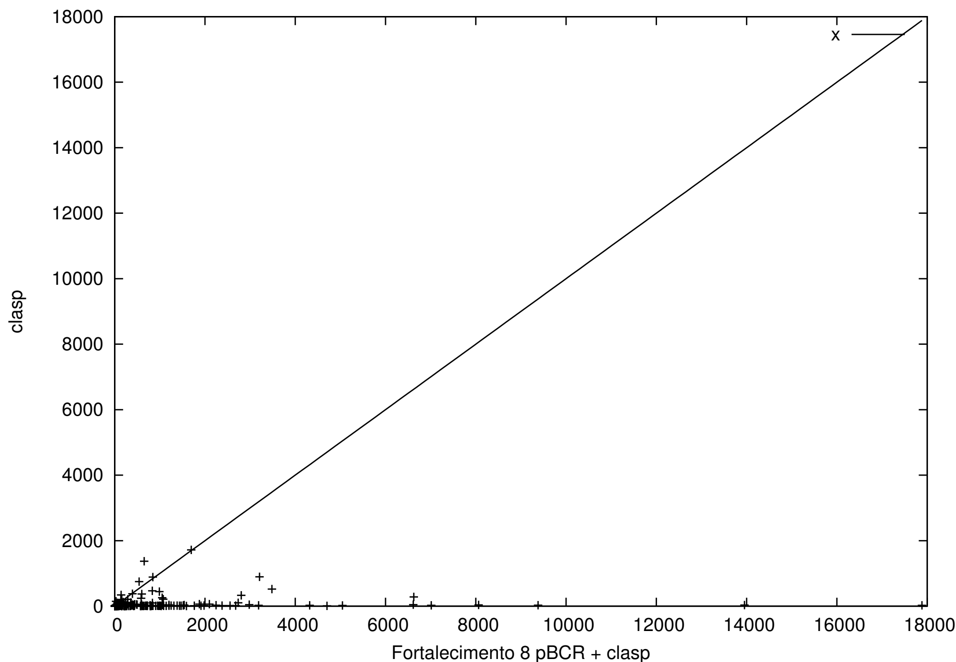

Figura 5.24: Comparação dos tempos de execução do CLASP nas fórmulas CNF originais e com fortalecimento de crescimento 8 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

fórmulas comumente resolvidas, 53 foram resolvidas mais rapidamente com as fórmulas fortalecidas, este número reduz para 27 quando se considerou o tempo utilizado pelo *pBCR*.

O tempo utilizado pelo CLASP nas 103 fórmulas foi de $13.881,6$ segundos nas fórmulas originais e de $16.837,4$ segundos nas fortalecidas. Com o tempo do *pBCR* foi de $17.315,3$ segundos. A figura 5.25 mostra a comparação de tempo entre estas duas configurações, ficando no eixo x a configuração com as fórmulas fortalecidas.

No fortalecimento 4 tivemos 103 fórmulas resolvidas, sendo 101 em conjunto com as fórmulas originais. O CLASP resolveu 3 fórmulas que com o fortalecimento não conseguiu resolver e 2 fórmulas foram resolvidas com o fortalecimento mas não sem modificação. O tempo total de processamento do CLASP nestas fórmulas foi de $10.619,6$ segundos nas fórmulas originais e de $13.927,9$ segundos nas fortalecidas, quando se considerou o tempo de processamento do *pBCR* o tempo ficou em $25.211,1$ segundos.

Com as fórmulas fortalecidas o CLASP foi mais rápido em 48 das 101 resolvidas

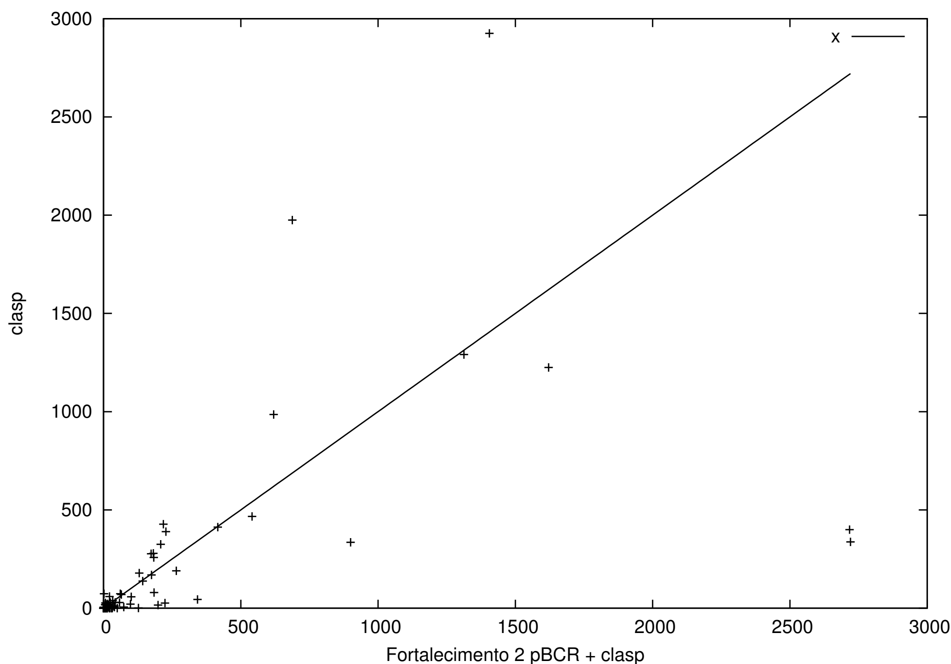

Figura 5.25: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 2 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

mutuamente, e em apenas 22 quando se considerou o tempo de processamento do *pBCR*. A figura 5.26 mostra a comparação de tempo entre estas duas configurações, ficando no eixo x a configuração com as fórmulas fortalecidas.

Finalmente para o fortalecimento 8, tivemos que 103 fórmulas foram resolvidas pelo CLASP, ficando 101 resolvidas mutuamente com as fórmulas originais. Destas, 3 foram resolvidas com as fórmulas originais e não foram com as fortalecidas e 2 foramo resolvidas com as fórmulas fortalecidas e foram são com as originais.

Dentre as fórmulas mutuamente resolvidas tivemos que o CLASP foi mais rápido em 47 fórmulas fortalecidas, no entanto, este valor foi reduzido para 21 quando se considerou o tempo de processamento do *pBCR*. A figura 5.27 mostra a comparação de tempo entre estas duas configurações, ficando no eixo x a configuração com as fórmulas fortalecidas.

Otimização

Nas 650 fórmulas de otimização o *pBCR* utilizou $2.176,62$ segundos de processamento para efetuar o fortalecimento de crescimento 2 e $2.191,04$ para o fortalecimento 4. Como

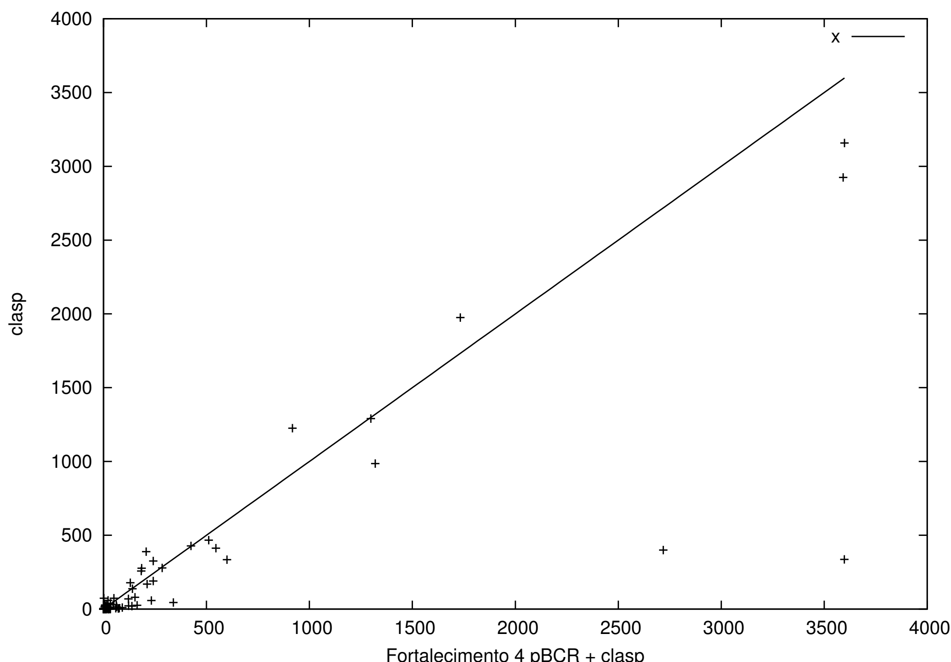

Figura 5.26: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 4 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

dito anteriormente, não foi efetuado o fortalecimento 8 para estas fórmulas por elas não cresceram mais de 4 literais por restrição na configuração usada.

Para este experimento vamos analisamos o CLASP e o SCIP de forma independentes.

CLASP

O CLASP conseguiu achar a solução ótima em apenas 70 fórmulas fortalecidas 2, e em 79 quando não se modificou. São 19 fórmulas em que a solução ótima foi encontrada sem modificar a fórmula, porém elas não foram encontradas nas fórmulas fortalecidas. Em 10 o CLASP encontrou o ótimo na fórmula fortalecida, mas não na original, ficando apenas 60 fórmulas resolvidas mutuamente entre as configurações.

Com as fórmulas fortalecidas o CLASP encontrou o ótimo em $9.664, 27$ segundos nas 60 fórmulas, e foram utilizados $13.216, 3$ segundos nas fórmulas originais. Uma redução de $26, 8\%$ nestas fórmulas. Apesar desta redução, com as fórmulas fortalecidas, o CLASP foi mais rápido em 27, e quando se considerou o tempo do *pBCR* foi reduzido para 21 fórmulas. A figura 5.28 mostra a comparação de tempo entre estas duas configurações,

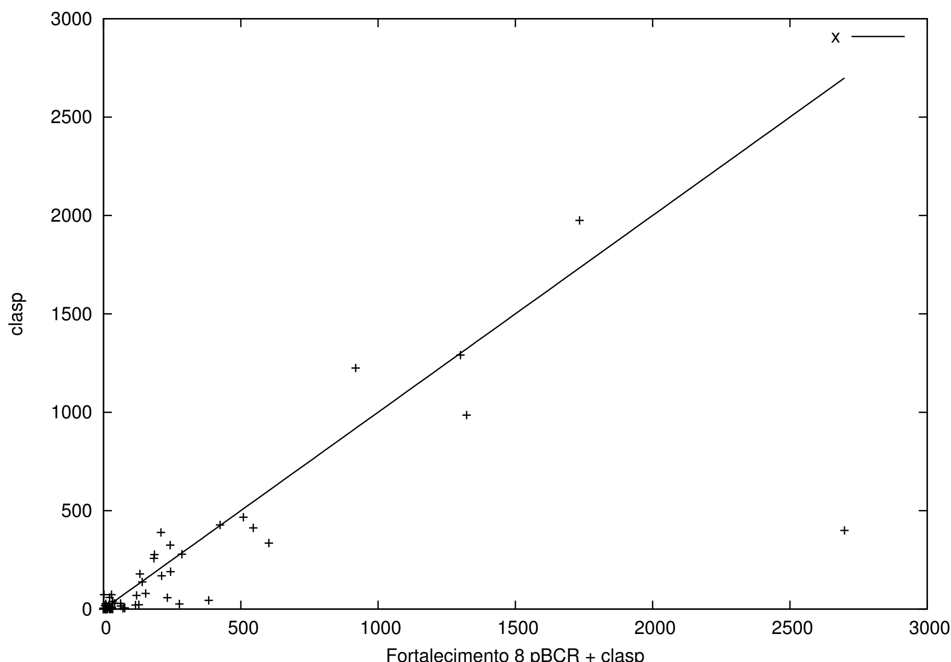

Figura 5.27: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 8 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

ficando no eixo x a configuração com as fórmulas fortalecidas.

Já para o fortalecimento 4, o CLASP conseguiu encontrar o ótimo em 75 fórmulas. São fórmulas que ele resolveu fortalecido que não são com as fórmulas originais e foram 14 que não foram resolvidas fortalecidas mas que o foram nas originais. São ao todo 65 fórmulas mutuamente resolvidas.

O tempo do CLASP para encontrar a solução ótima nas 65 fórmulas foi de $10.720,3$ segundos, contra $13.594,5$ segundos nas fórmulas originais. Quando se considerou o tempo do *pBCR* o tempo total foi de $11.216,5$ segundos nas fórmulas fortalecidas. A redução neste experimento é de $17,49\%$, considerando o tempo do *pBCR*. Ainda sim o CLASP nas fórmulas fortalecidas obteve melhor tempo em 26 delas, e em 22 quando se considerou o tempo de pré-processamento. A figura 5.29 mostra a comparação de tempo entre estas duas configurações, ficando no eixo x a configuração com as fórmulas fortalecidas.

SCIP

O SCIP conseguiu encontrar o ótimo em 234 fórmulas fortalecidas 2, enquanto que

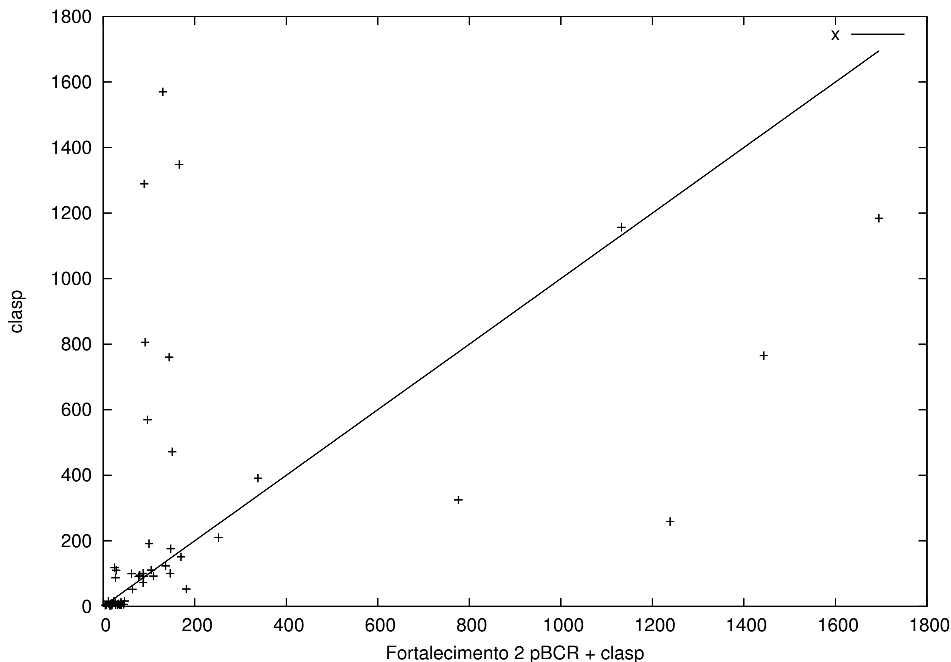

Figura 5.28: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 2 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

encontrou em 239 das fórmulas originais. Destas fórmulas 22 não foram resolvidas quando fortalecidas, e 17 só foram resolvidas com o fortalecimento 4. Deste modo obtivemos 217 fórmulas resolvidas mutuamente entre as configurações.

O SCIP utilizou $93.222,7$ segundos para achar o ótimo nas 217 fórmulas, e 120.227 segundos nas fórmulas originais. Quando se considerou o tempo do *pBCR* tivemos $93.977,7$ segundos para as fórmulas fortalecidas, uma redução de $21,83\%$ no tempo de execução. O SCIP foi mais rápido em 173 fórmulas, e este número se manteve quando foi considerado o tempo de execução do *pBCR*. A figura 5.30 mostra a comparação de tempo entre estas duas configurações, ficando no eixo x a configuração com as fórmulas fortalecidas.

Para o fortalecimento 4 o SCIP encontrou o ótimo em 238 fórmulas. Do conjunto, 24 não foram resolvidas pelo SCIP sem modificação na fórmula, e 25 não foram resolvidas com o fortalecimento, finalizando com 214 fórmulas mutuamente resolvidas por ambas configurações.

O tempo utilizado pelo SCIP nas fórmulas originais foi de 115.698 segundos, contra

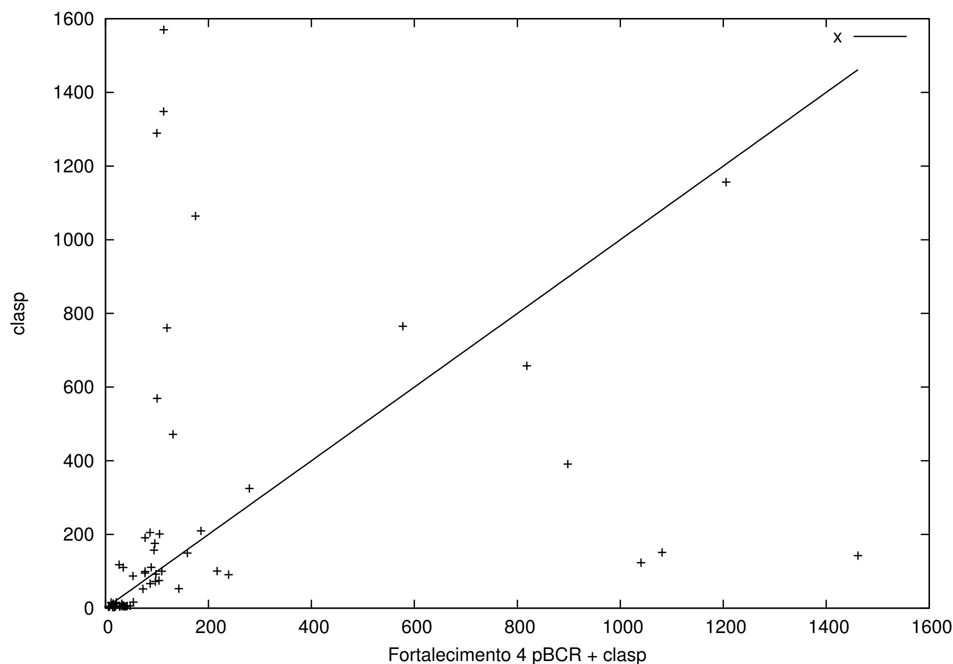

Figura 5.29: Comparação dos tempos de execução do CLASP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 4 pelo *pBCR*. O gráfico mostra o tempo em segundos do CLASP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

90.547, 1 segundos nas fortalecidas e 91.276, 5 segundos quando foi considerado o tempo do *pBCR*. A redução para esta configuração foi de 21, 10%, mesmo considerando o tempo de execução do pré-processamento. O SCIP foi mais rápido em 176 fórmulas fortalecidas e, em 175 quando foi considerado o tempo do *pBCR*. A figura 5.31 mostra a comparação de tempo entre estas duas configurações, ficando no eixo x a configuração com as fórmulas fortalecidas.

5.4 pBCR completo

Para o experimento do *pBCR* completo, habilitamos todas as etapas e continuamos variando o limite de fortalecimento por restrição em 2,4 e 8 literais.

Considerando todas as variações de etapas e as fórmulas originais tivemos 17.378 fórmulas testadas. Com estes experimentos pudemos tirar algumas conclusões. Abaixo há um panorama do desempenho dos resolvedores nas fórmulas originais e pré-processadas pelo *pBCR* contemplando todas suas variações para as fórmulas em CNF, pseudo-Boolean.

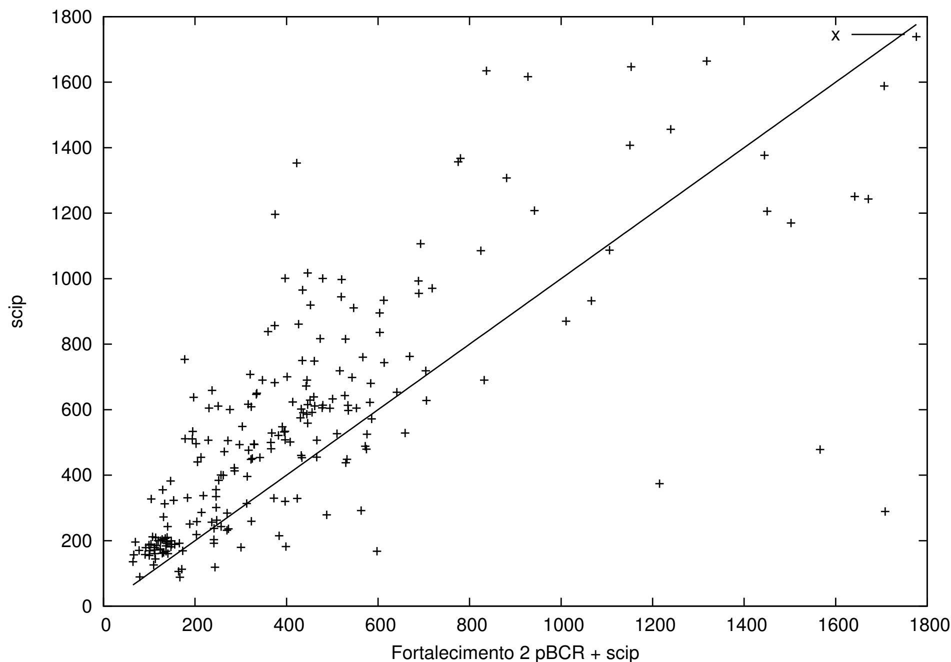

Figura 5.30: Comparação dos tempos de execução do SCIP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 2 pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

Em seguida foi analisado o que aconteceu com as fórmulas experimentadas.

5.4.1 Fórmulas CNF

Para as fórmulas CNF utilizadas no experimento, o GLUCOSE foi o melhor resolvedor, pois foi o único que resolveu todas as fórmulas dentro do limite de tempo estipulado de 3.600 segundos.

O LINGELING possuiu um desempenho superior ao CLASP em CNF tanto nas fórmulas originais como nas fórmulas sondadas. Quando comparado com o CLASP executando as fórmulas CNF escritas em pseudo-Boolean, o seu desempenho continuou inferior ao CLASP em CNF. Em nenhuma configuração o CLASP pseudo-Booleano conseguiu bater o LINGELING em quantidade de fórmulas resolvidas.

No conjunto das etapas do *pBCR*, a configuração com sondagem e aprendizado obteve melhor resultado com o CLASP em pseudo-Boolean, com 534 fórmulas resolvidas, mas ainda perdeu do CLASP em CNF, com o *pBCR* efetuando apenas a sondagem, pois obteve

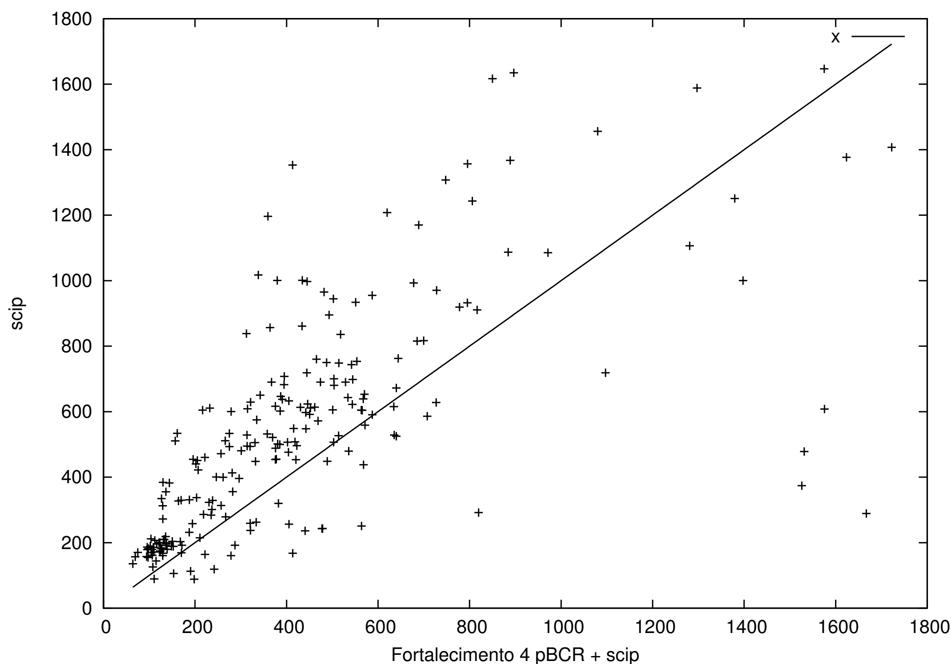

Figura 5.31: Comparação dos tempos de execução do SCIP nas fórmulas pseudo-Booleanas originais e com fortalecimento de crescimento 4 pelo *pBCR*. O gráfico mostra o tempo em segundos do SCIP com as fórmulas originais no eixo y contra o CLASP com as fórmulas fortalecidas no eixo x. A linha $f(x) = x$ é colocada como referência.

536 fórmulas resolvidas.

A tabela 5.5 mostra o comparativo da quantidade de fórmulas resolvidas em cada uma das configurações, considerando somente este quesito o GLUCOSE foi imbatível por ter resolvido todas as fórmulas.

pBCR	GLUCOSE	LINGELING	CLASP	CLASP PB
—	549	538	532	530
Sondagem	549	539	536	532
Aprendizado	—	—	—	534
Fortalecedor 2	—	—	—	502
Fortalecedor 4	—	—	—	499
Fortalecedor 8	—	—	—	494
completo 2	—	—	—	515
completo 4	—	—	—	516
completo 8	—	—	—	508

Tabela 5.5: Quantidade de fórmulas resolvidas pelos resolvedores GLUCOSE, LINGELING, CLASP e CLASP em pseudo-Boolean dentre as 549 fórmulas CNF

Ficou claro que o resolvedor pseudo-Boolean ainda precisa de alguns ajustes para se

igualar em desempenho aos resolvedores tradicionais em CNF, mas também foi identificado que, parte da ajuda gerada pelo *pBCR* na sondagem e no aprendizado permitiu o CLASP a obter uma maior quantidade de fórmulas resolvidas.

5.4.2 Fórmulas pseudo-Boolean

Decisão

Dentre as fórmulas de decisão, 2 foram provadas trivialmente insatisfatíveis pelo *pBCR* restando 114 fórmulas a serem resolvidas. O CLASP resolveu 104 delas e a melhor configuração foi com *pBCR* fazendo a sondagem e o fortalecimento de crescimento 2, sendo resolvidas 105 fórmulas. Já o SCIP conseguiu resolver apenas 29 fórmulas e nenhuma configuração do *pBCR* conseguiu auxiliar este resolvedor a melhorar a quantidade de fórmulas resolvidas. A tabela 5.6 apresenta a quantidade de fórmulas resolvidas pelo CLASP e SCIP em cada uma das configurações selecionadas.

pBCR	CLASP PB	SCIP
—	104	29
Sondagem	104	28
Aprendizado	102	28
Fortalecedor 2	105	26
Fortalecedor 4	103	25
Fortalecedor 8	103	25
completo 2	104	25
completo 4	102	26
completo 8	101	26

Tabela 5.6: Quantidade de fórmulas resolvidas pelo CLASP e SCIP dentre as 114 fórmulas pseudo-Booleanas

Embora o SCIP, um clássico resolvedor de programação linear, também consiga resolver fórmulas que não possuem funções objetivo, o mesmo não conseguiu se igualar ao CLASP neste tipos de fórmulas, ficando muito aquém ao esperado para as fórmulas em questão.

O CLASP obteve um desempenho muito superior, tendo conseguido resolver a maioria das fórmulas. Apenas a combinação da sondagem com o fortalecimento 2 auxiliou o

resolvedor a aumentar em uma fórmula resolvida. A adição de aprendizado e maiores fortalecimentos piorou o resultado do CLASP.

Otimização

Na otimização, o resolvedor SCIP foi superior ao CLASP. Neste resolvedor tivemos 239 fórmulas dentre as 650 com o ótimo encontrado no tempo limite estipulado. A melhor configuração foi efetuando apenas a sondagem, fazendo o número saltar para 254 fórmulas. Já com o CLASP, apenas 79 fórmulas foram encontradas o ótimo, e em nenhuma configuração conseguiu-se fazer o número de ótimos encontrados aumentar.

A tabela 5.7 apresenta o desempenho dos resolvedores. O SCIP obteve um melhor resultado com a sondagem executada pelo *pBCR* e somente na configuração com fortalecimento restrito a 4 literais que obteve um resultado semelhante ao SCIP sem qualquer pré-processamento do *pBCR*.

pBCR	CLASP PB	SCIP
—	79	239
Sondagem	78	254
Aprendizado	76	212
Fortalecedor 2	70	234
Fortalecedor 4	75	238
completo 2	70	218
completo 4	75	216

Tabela 5.7: Quantidade de fórmulas em que o CLASP e o SCIP encontraram a solução ótima dentre as 650 fórmulas de otimização pseudo-Booleanas

5.4.3 Mudanças nas fórmulas

Dentre as diversas etapas do *pBCR*, a da sondagem e a da subjugação, aliada com o fortalecimento, algumas restrições foram eliminadas, gerando fórmulas menores em número de restrições. Isto auxiliou os resolvedores, pois houve menos restrições a serem analisadas.

Primeiramente, observou-se que as fórmulas em CNF possuem ao todo 2.628.324.564 restrições e após a sondagem o número reduz para 2.409.787.989, sendo uma redução de 8% no número global de restrições. Quando se habilitou o fortalecimento, o número global de restrições ficou em 2.399.600.617, 2.395.409.662 e 2.386.239.679, para os limites de

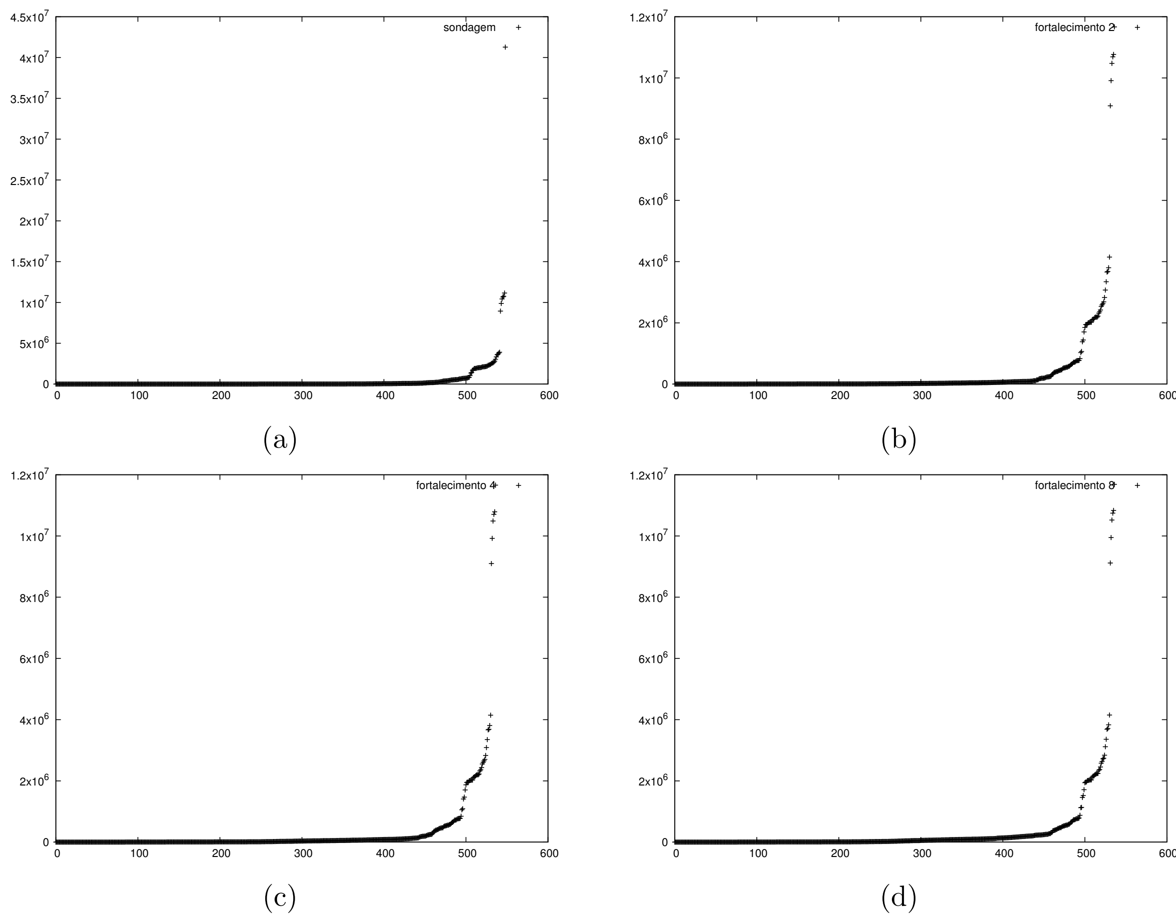

Figura 5.32: Restrições eliminadas nas 549 fórmulas em CNF nas configurações de sondagem, fortalecimento de crescrimento 2, 4 e 8 do *pBCR*

crescimento de 2, 4 e 8, causando uma redução de $8,70\%$,$8,86\%$ e $9,21\%$, respectivamente. A figura 5.32 mostra a redução crescente das 549 fórmulas em CNF nas configurações de sondagem e fortalecimento de crescimento 2,4 e 8 do *pBCR*.

Para estas mesmas fórmulas, nas originais haviam 6.461 variáveis que já possuíam valoração fixada. Com a sondagem este número aumentou para 1.485.297. Para as configurações de fortalecimento as variáveis fixadas mudaram para 1.392.934, 1.411.307 e 1.420.099. A variação reduzida no fortalecimento 2 se deu pelo fato de que 12 fórmulas foram identificadas como trivialmente insatisfatíveis, e as suas variáveis não foram contabilizadas. O mesmo ocorreu com o fortalecimento 4 e 8, porém as reduções adicionais que aconteceram nessas configurações ultrapassaram os valores dessas 12 fórmulas. A figura 5.33 apresenta a quantidade de variáveis eliminadas em cada uma das 549 fórmulas, em cada uma das configurações do *pBCR*, para o gráfico os valores foram ordenados do

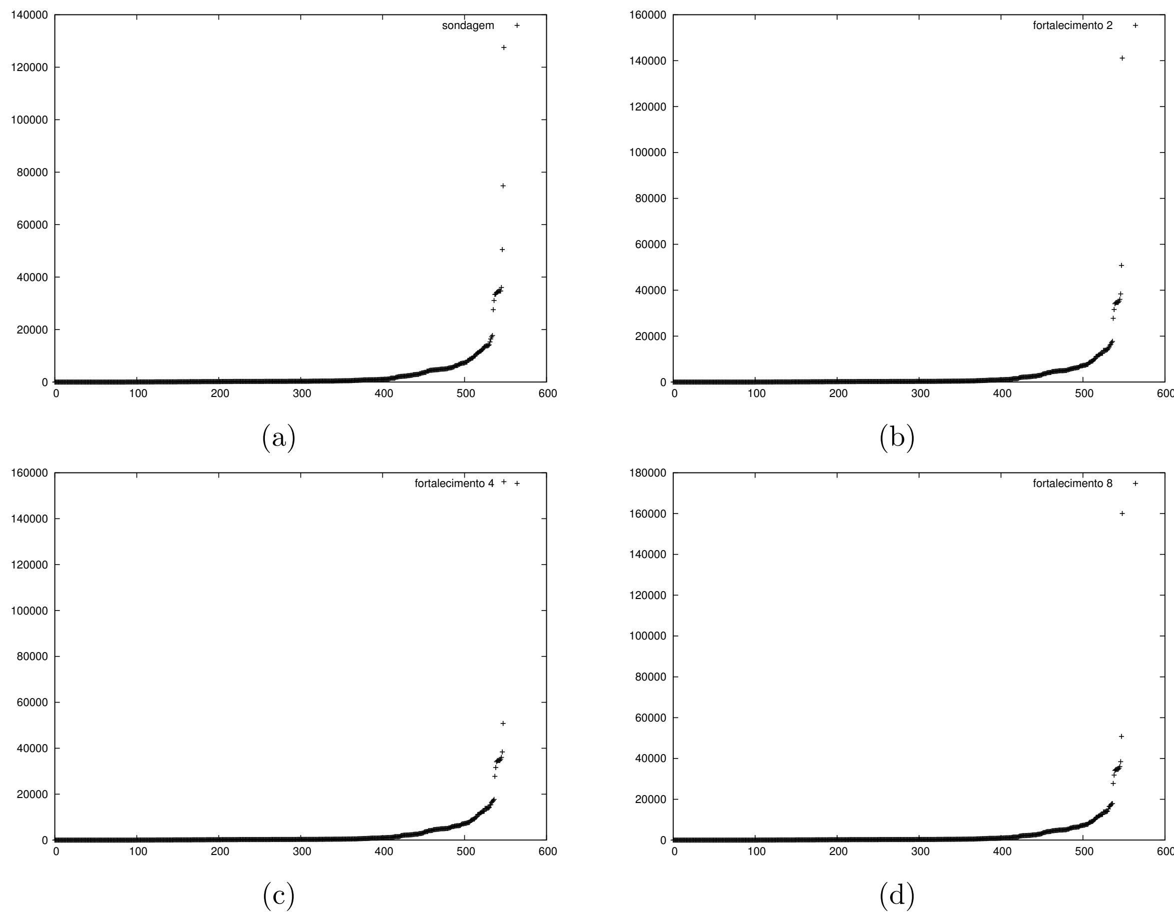

Figura 5.33: Variáveis eliminadas nas 549 fórmulas em CNF nas configurações de sondagem, fortalecimento de crescrimento 2, 4 e 8 do *pBCR*

menor para o maior.

Para as fórmulas de decisão pseudo-Booleanas, o índice foi de $45,96\%$ de redução para a sondagem. Quando aplicou-se o fortalecimento de crescimento 2,4 e 8 tivemos reduções de, respectivamente, $48,36\%$, $48,50\%$ e $48,57\%$. A quantidade de restrições inicialmente era de 15.302.218 e reduziu para 8.267.814 com a sondagem. Com os fortalecimentos foram para 7.901.479, 7.879.289, 7.869.871. A figura 5.34 apresenta a redução das restrições das 114 fórmulas com as configurações do *pBCR*.

Ainda nas fórmulas de decisão pseudo-Booleanas, o número de variáveis que foram eliminadas cresceu, nas fórmulas originais haviam 491.286 variáveis decididas, com a sondagem este número cresceu para 1.769.855 e para as configurações de fortalecimento temos 1.776.255, 1.776.259 e 1.776.259 para os fortalecimentos de crescimento 2,4 e 8. A figura 5.35 mostra a evolução de remoção de variáveis nas fórmulas para estas confi-

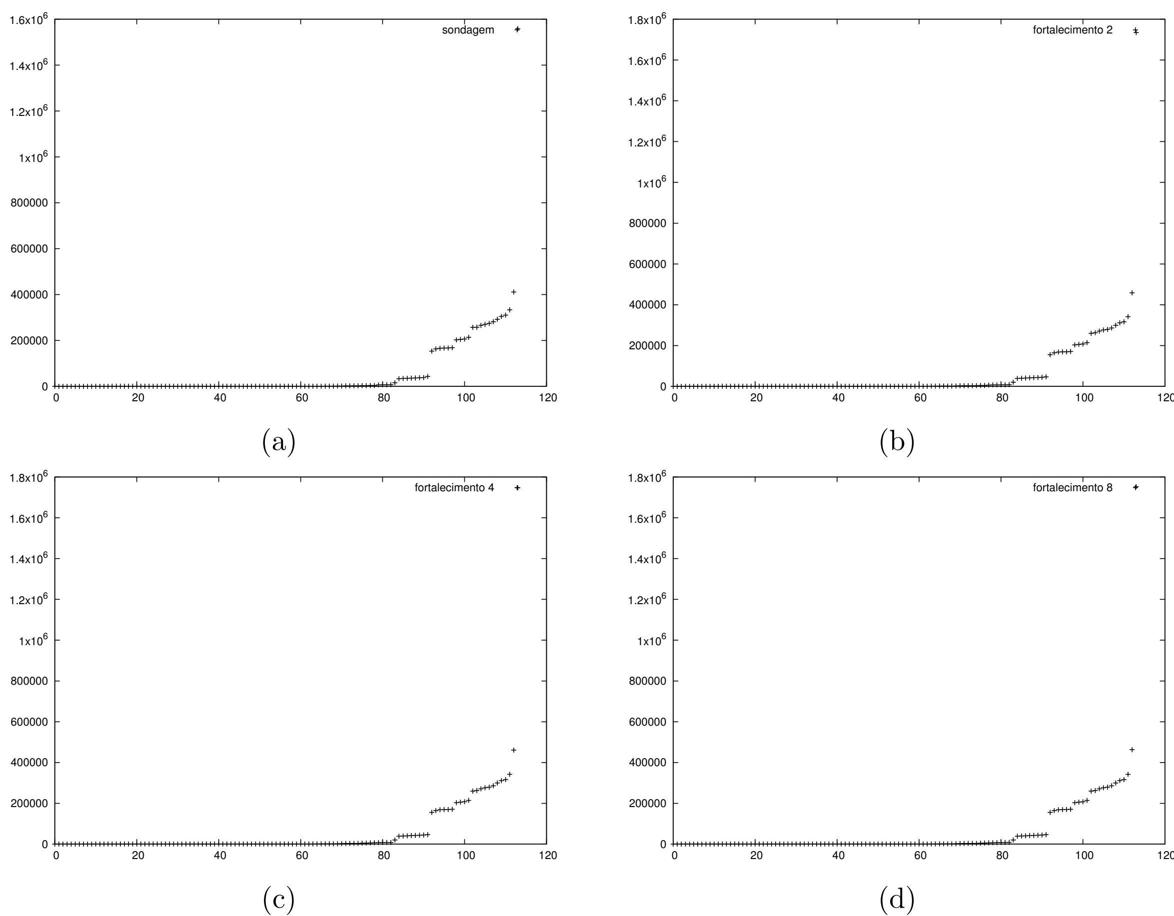

Figura 5.34: Restrições eliminadas nas 114 fórmulas pseudo-Booleanas nas configurações de sondagem, fortalecimento de crescimento 2, 4 e 8 do *pBCR*

gurações.

Para as fórmulas de otimização pseudo-Booleana tivemos uma redução de $20,77\%$ nas restrições com a sondagem, $20,99\%$ para o fortalecimento 2, e 21% para o fortalecimento 4. Os valores de todas as restrições passaram de 33.905.146 originalmente para 26.860.050 com a sondagem, 26.785.543 e 26.783.374 para o fortalecimento 2 e 4. A figura 5.36 apresentou a quantidade de restrições removidas das 650 fórmulas, ordenadas da menor para a maior.

Já para as variáveis decididas, originalmente tivemos 380.394 e quando sondadas tivemos ao todo 8.315.644 variáveis. Para o fortalecimento 2 e 4 tivemos o mesmo valor de 8.316.354. A figura 5.37 apresenta a evolução da remoção de variáveis nas fórmulas de otimização para as configurações consideradas.

Também avaliamos o impacto da aridade da fórmula, ou seja, a quantidade de literais

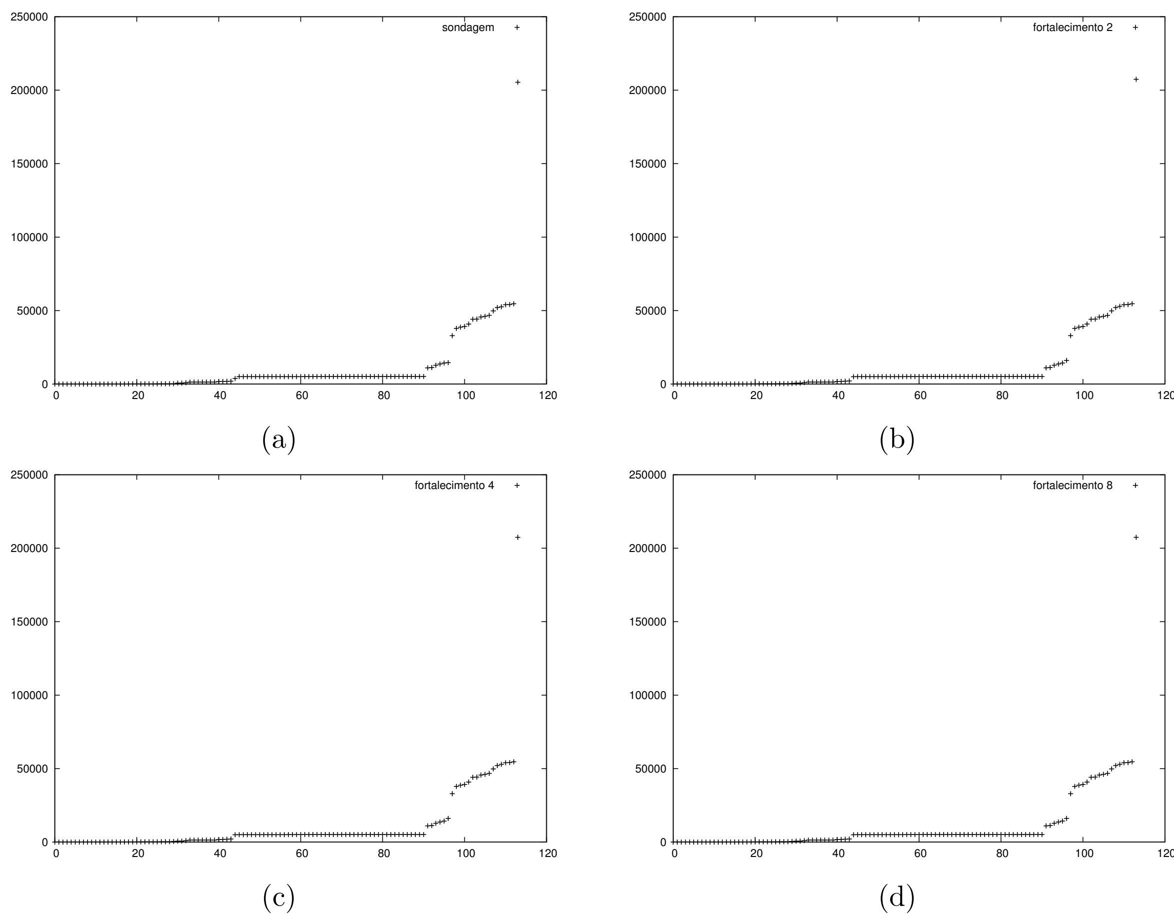

Figura 5.35: Variáveis eliminadas nas 114 fórmulas pseudo-Booleanas nas configurações de sondagem, fortalecimento de crescimento 2, 4 e 8 do *pBCR*

por restrição. Quanto maior, a aridade da fórmula representa uma oneração ao mecanismo de propagação, principalmente em pseudo-Boolean, que pode ter que fazer uma composição de literais a cada valoração que não auxilia na restrição. E mesmo em CNF, uma aridade grande pode fazer com que o resolvedor valore muitas variáveis até que identifique um problema na valoração parcial, pois existem muitas possibilidades de literais a serem decididos, até que a cláusula fique em estado crítico de valoração.

Nas fórmulas em CNF tivemos um impacto elevado na aridade das fórmulas. No geral a aridade das fórmulas é de $2,02$ literais por cláusula, a sondagem mantém esta aridade e, no fortalecimento tivemos o aumento da aridade para $3,99$, $5,94$ e $9,83$, respectivamente para os crescimentos 2, 4 e 8.

O impacto do fortalecimento na aridade nas fórmulas pseudo-Booleanas foi bem baixo. Nas fórmulas de decisão, a aridade média das fórmulas originais é de $2,83$ literais por

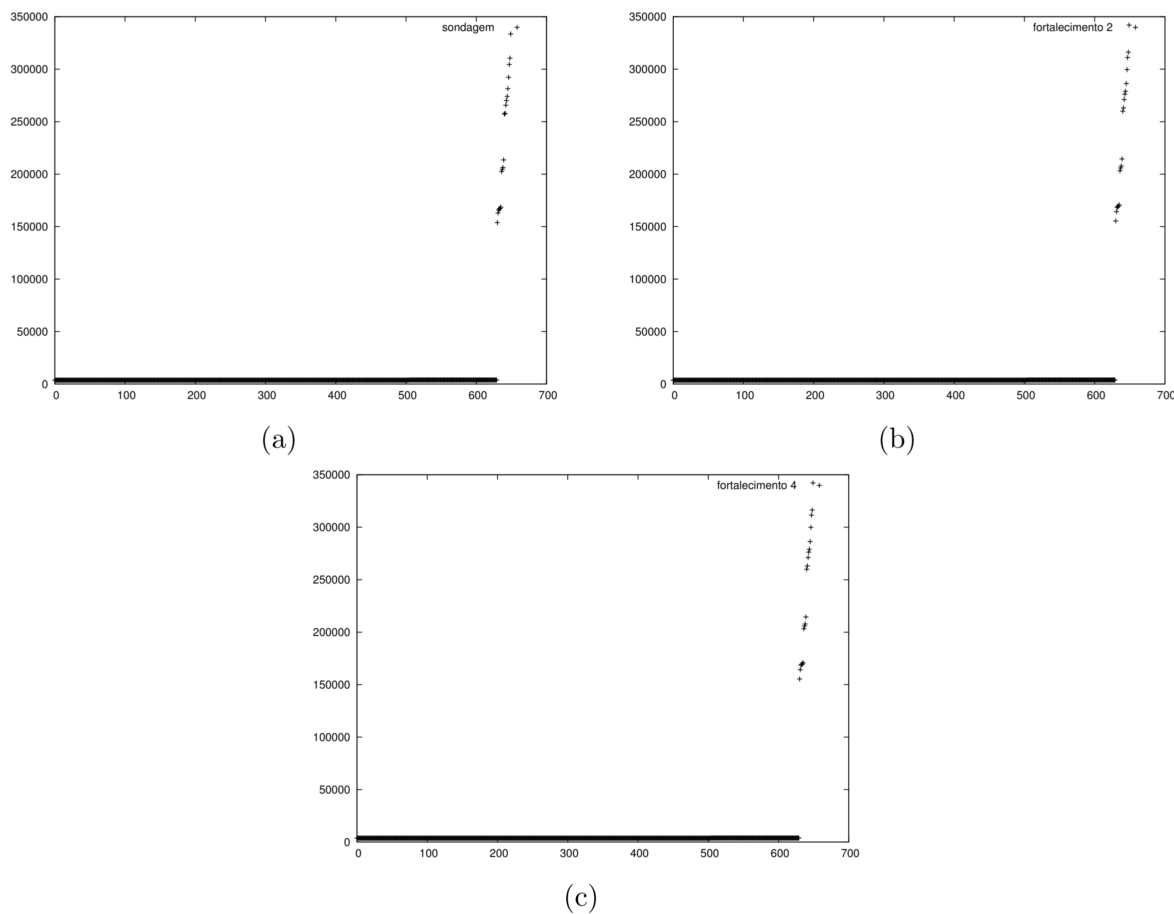

Figura 5.36: Restrições eliminadas nas 650 fórmulas pseudo-Booleanas de otimização nas configurações de sondagem, fortalecimento de crescimento 2 e 4 do *pBCR*

restrição, caindo para 2,73 com a sondagem e, subindo para 3,26, 3,33 e 3,34 para as configurações de fortalecimento.

A aridade, para as fórmulas de otimização pseudo-Booleanas, possuem uma média de 4,92 literais por restrição, subindo para 5,26 na sondagem e com 5,36 e 5,37 nas configurações de fortalecimento.

5.5 Considerações Finais

Neste capítulo, pudemos analisar e entender os impactos de cada uma das etapas do *pBCR* nos resolvedores que recebem a fórmula pré-processada. Como a hipótese desta tese, podemos afirmar que existe um impacto nos resolvedores. O impacto às vezes se reflete em um tempo melhor de decisão ou, otimização da fórmula, e em outros casos reflete em um pior tempo.

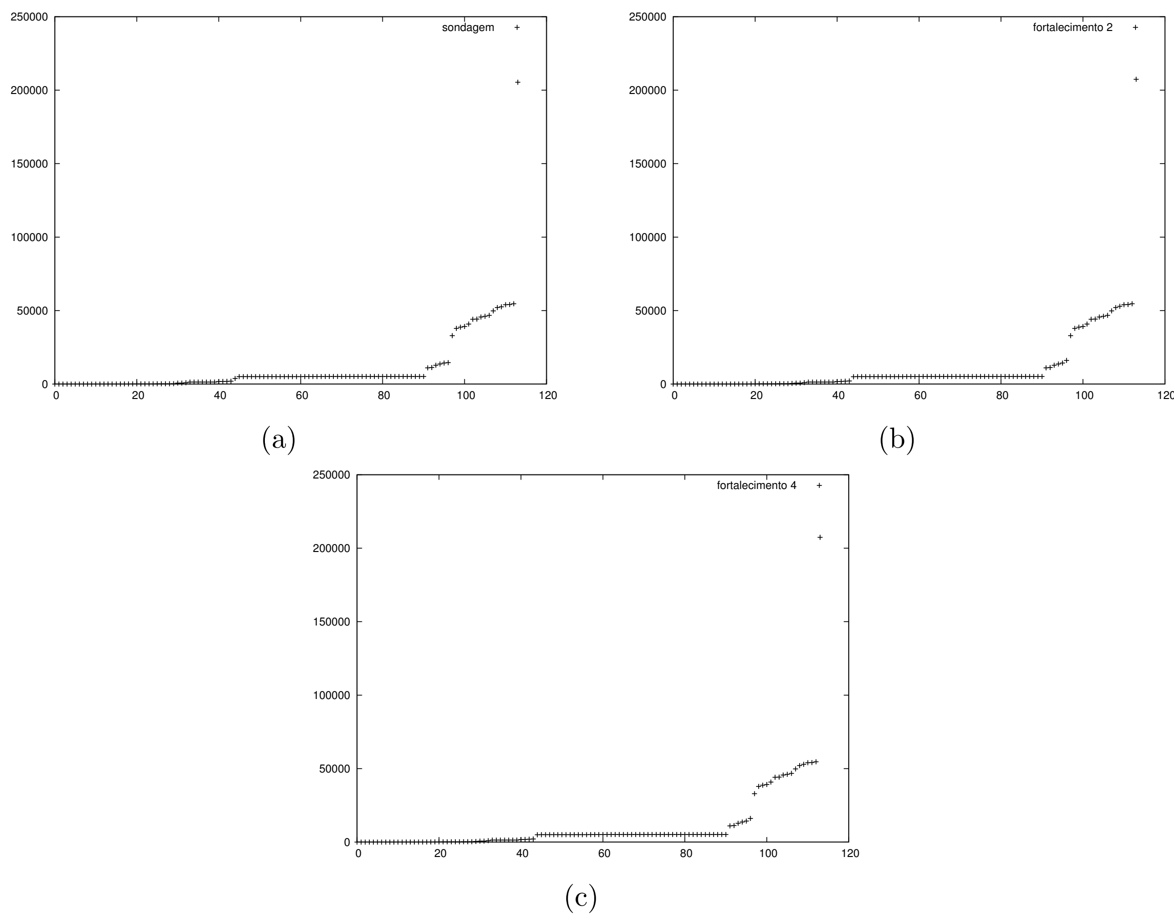

Figura 5.37: Variáveis eliminadas nas 650 fórmulas pseudo-Booleanas de otimização nas configurações de sondagem, fortalecimento de crescimento 2 e 4 do *pBCR*

O método de pré-processamento que causou o melhor impacto, ou seja, que redzuiu o tempo de processamento do resolvedor, foi o método da sondagem. A sondagem é o método que faz parte do processamento do resolvedor, e por conseguir eliminar variáveis o espaço de busca para o resolvedor é reduzido.

O aprendizado não causa um impacto positivo muito claro, o principal motivo é que o aprendizado injetado na fórmula é apenas um processo de resolução, que o resolvedor já faz, na maior parte das fórmulas a informação agregada apenas aumenta processamento para o resolvedor.

Dentre o conjuntos de fórmulas utilizado nesta tese, identificamos que o *pBCR* realmente modifica as fórmulas, reduzindo, em alguns casos, em 3 vezes o tamanho da fórmula, gerando uma fórmula com uma representação mais forte, permitindo que o resolvedor tenha menos restrições para analisar a cada passo e, ainda, impacta no tamanho

em bytes do arquivo da fórmula.

O impacto negativo do fortalecimento nas fórmulas em CNF, principalmente das oriundas dos problema de planejamento, não eram o esperado pelos experimentos iniciais de Dixon [20] em sua tese.Pois muitos problemas de planejamento podem possuir o mesmo potencial de fortalecimento que a modelagem do princípio da casa dos pombos. E, de fato, as fórmulas possuem um grande impacto com o fortalecimento. Embora diversas fórmulas tenham sido fortalecidas e tido diversas restrições subjugadas, o resolvedor utilizado ficou menos eficiente. Muito se deve ao fato das restrições terem aumentado de tamanho, tornando o processo de análise por restrição, mais lento no momento de escolha do próximo literal vigiado.

A análise mais lenta das restrições, conforme acontece o fortalecimento, foi algo que impactou o tempo do *pBCR*. Por isso a utilização dos mesmos literais da sondagem foi utilizada para o fortalecimento, pois o efeito na fórmula é mais brando, e não se precisa trocar com mais frequência o literal vigiado.

Por fim, avaliamos o impacto do *pBCR* como positivo. Uma vez que, algumas fórmulas que não são resolvidas sem qualquer pré-processamento, o são quando pré-processadas pelo nosso algoritmo.

Apesar de algumas fórmulas sofrerem o impacto negativo do pré-processamento, sabemos que elas podem ainda ser resolvidas sem a utilização do *pBCR*. Então, considerando um resolvedor portfólio, um resolvedor que execute diversos algoritmos e estratégias, como é o caso do SATzilla [69], podemos afirmar que a utilização do *pBCR* seria positiva.

CAPÍTULO 6

PSEUDO-BOOLEAN APLICADO À CONSOLIDAÇÃO DE MÁQUINAS VIRTUAIS

Computação em Nuvem é um paradigma recente e crescente de computação distribuída, que oferece recursos virtualizados e serviços na Internet [43, 2]. Com o uso deste paradigma é possível oferecer um arcabouço de recursos facilmente utilizáveis e acessíveis. Esses recursos podem ser reconfigurados dinamicamente para se ajustar às necessidades de utilização, tais como carga de processamento. Esse arcabouço é explorado tipicamente pelo modelo de *pay-per-use* onde é garantido que o recurso pedido é oferecido pelo provedor da infraestrutura [65].

Um dos modelos de serviço oferecido na Nuvem é *Infrastructure-as-a-Service (IaaS)* onde os recursos virtualizados são providos por meio de uma máquina virtual (VM). Com as máquinas virtuais os usuários passam a ter um ambiente de execução isolado e personalizado para suas aplicações. Uma VM também utiliza recursos virtualizados como CPU, RAM, rede e dispositivos de armazenamento.

Muitos provedores de Cloud utilizam um grande *data center* para oferecer IaaS. Os *data centers* possuem uma grande quantidade de recursos físicos (servidores, discos, redes cabeadas). Infelizmente, muito dos grandes *data center* possuem um uso de seus recursos muito baixo, variando entre 5% e 10% de sua capacidade total. Para maximizar a utilização de recursos físicos por meio dos recursos virtualizados, um provedor Cloud IaaS pode aplicar a técnica de consolidação de servidores [51, 67, 27] para a relocação de VM em servidores físicos. Essa técnica de consolidação também é denotada de *VM Consolidation.*

A consolidação pode aumentar a carga dos servidores que pode variar entre 50% a 85%. Com isso os *hardware* modernos conseguem operar mais eficientemente em termos de consumo de energia [32]. Em alguns casos, a consolidação pode economizar 75% de

energia [14]. A realocação de recursos virtualizados permite o desligamento de servidores físicos, reduzindo custos de refrigeração, administração de *hardware* e consumo de energia.

Com o objetivo de maximizar o uso de um *data center*, uma consolidação de VM ótima tem sido um tópico de pesquisa em Cloud Computing. Existem trabalhos que utilizam formulação em *Programação Linear* ou algoritmos distribuídos para garantir a utilização ótima dos recursos [51, 67, 27, 11].

Diferentemente das abordagens acima, uma maneira de formular o problema utilizando restrições pseudo-Booleanas é trabalhada desde 2012. A primeira formulação foi publicada [56] no IBERAMIA 2012. Resultou em um modelo bastante enxuto, porém muito complicado para os resolvedores mais modernos solucionarem. O segundo trabalho publicado [55] no BRACIS 2013, trouxe uma formulação muito mais próxima do princípio da casa dos pombos, que se mostrou ser menos enxuta (com mais restrições, porém com metade das variáveis). Porém, muito mais simples de se resolver, quando aplicado o fortalecimento das restrições. Esta nova formulação foi chamada de *PBFVMC*.

A característica mais importante da segunda formulação, *PBFVMC*, é a possibilidade de aplicar o *pBCR* e obter uma fórmula com menos restrições. Apresentaremos neste capítulo ambas as formulações com o intuito de entender que tipos de alterações foram feitas na primeira modificação para gerar a PBFVMC, que pode ser aplicada com o método proposto nesta tese.

As descrições das duas versões da formulação seguem nas próximas seções, bem como uma avaliação experimental.

6.1 A primeira versão

Para o problema de consolidação de máquinas virtuais deve-se colocar K VMs $\{vm_1 \ldots vm_K\}$ em execução dentro de N *hardware* $\{hw_1 \ldots hw_N\}$ enquanto minimiza o número total de *hardware* ligados. Cada VM vm_i possui, associada às suas necessidades, a quantidade de núcleos de processamento (VCPU) e, a quantidade de memória RAM (VRAM), enquanto que cada *hardware* hw_j possui a quantidade de recursos disponível, em processamento (CPU) e memória (RAM).

Esta formulação possui 6 tipos de restrições, que totalizam $(2 \times N + 2 \times N \times K)$ variáveis e $(2+2\times N+K)$ restrições, onde N e K representam, respectivamente, o número de *hardware* disponível e a quantidade de máquinas virtuais. A formulação é bastante compacta, porém gera instâncias difíceis de serem solucionadas por um resolvedor pseudo-Booleano, já que a maior fórmula que foi provada como satisfatível possuía apenas 128 *hardware* com 1277 virtuais e levou 14400 segundos. Além disse a maior fórmula que se pôde provar um resultado ótimo foi com 32 *hardware* e 98 virtuais.

Esta formulação é feita da seguinte forma:

Para se fazer as restrições pseudo-Boolanas, cada *hardware* possui duas variáveis, uma que relaciona o *hardware* hw_i com a quantidade de RAM hw_i^r e outra, que relaciona a quantidade de CPU hw_i^p. Para cada *hardware*, uma máquina virtual (VM) possui 2 variáveis. Uma para relacionar a quantidade necessária de VRAM vm_j^r do *hardware* hw_i^r, sendo denotada como $vm_j^{r \cdot hw_i}$. A outra variável relaciona a quantidade de VCPU necessária VCPU vm_j^p com a quantidade de CPU disponível do *hardware* hw_i^p, denotada por $vm_j^{p \cdot hw_i}$. Sendo assim, a quantidade de variáveis para representar as máquinas virtuais é de $2 \times N$.

Um *hardware* é considerado LIGADO quando as variáveis hw_i^r e hw_i^p são *Verdadeiras*, senão ele está DESLIGADO.

$$minimize : \sum_{i=1}^{N} hw_i \tag{6.1}$$

$$\sum_{i=1}^{N} R_{hw_i} \cdot hw_i^r \geq \sum_{j=1}^{K} R_{vm_j} \tag{6.2}$$

$$\sum_{i=1}^{N} P_{hw_i} \cdot hw_i^p \geq \sum_{j=1}^{K} P_{vm_j} \tag{6.3}$$

$$\forall\, i \in 1..N \left(\sum_{j=1}^{K} R_{vm_j} \cdot vm_j^{r \cdot hw_i} \leq R_{hw_i} \right) \tag{6.4}$$

$$\forall\, i \in 1..N \left(\sum_{j=1}^{K} P_{vm_j} \cdot vm_j^{p \cdot hw_i} \leq P_{hw_i} \right) \tag{6.5}$$

$$\forall j \in 1..K \left(\sum_{i=1}^{N} vm_j^{p \cdot hw_i} \cdot vm_j^{r \cdot hw_i} \cdot hw_i^p \cdot hw_i^r = 1 \right) \tag{6.6}$$

A função objetivo é a soma dos *hardware* LIGADOS.

As restrições (6.2) e (6.3) garantem que a quantidade necessária de recursos dos *hardware* ligados são o suficiente para alocar todas as virtuais. Para definir o limite superior dos *hardware*, as restrições (6.4) e (6.5) definem a quantidade máxima de recursos que cada *hardware* pode prover. Finalmente a restrição (6.6) garante que a máquina virtual está sendo executado em exatamente um *hardware* e, dada a sua estrutura não-linear, é implícito que se uma máquina virtual está rodando em um *hardware*, este deve estar ligado.

6.2 PBFVMC

A formulação descrita na seção anterior é comparável à formulação de problemas de *Bin Packing*, especialmente a restrição (6.6). Por outro lado a PBFVMC se assemelha com uma formulação do princípio da casa dos pombos, com uma casa especial (o *hardware*) que pode armazenar mais de um pombo (máquina virtual) limitando-se pela quantidade de recursos disponíveis (RAM e CPU).

6.2.1 O modelo proposto

A maioria da estrutura definida na seção anterior é mantida. Todas as variáveis pseudo-Booleanas que foram associadas para a RAM (e VRAM) e CPU (e VCPU), anteriormente chamadas de hw_i^r (e vm_j^r) e hw_i^p (e vm_j^p) foram colapsadas para hw_i para *hardware* e vm_j para máquinas virtuais.

Dado isto, a nova formulação possui as seguintes variáveis:

- N : Número total de *hardware* disponível (hw);

- K : Número total de máquinas virtuais (VM);

- hw_i : Hardware $i \in N$;

- $vm_j^{hw_i}$: Máquina Virtual $j \in K$ executando em hw_i;

Um *hardware* é considerado LIGADO quando hw_i é *Verdadeiro*, senão é considerado DESLIGADO.

Apesar de não existir separação entre as variáveis que relacionam a quantidade de RAM e processamento de cada máquina virtual, ainda temos para cada máquina virtual uma variável que a relaciona com um *hardware*. Com isso temos $(N + N \times K)$ variáveis.

As restrições da formulação PBFVMC são:

$$minimize : \sum_{i=1}^{N} hw_i \tag{6.7}$$

$$\sum_{i=1}^{N} R_{hw_i} \cdot hw_i \geq \sum_{j=1}^{K} R_{vm_j} \tag{6.8}$$

$$\sum_{i=1}^{N} P_{hw_i} \cdot hw_i \geq \sum_{j=1}^{K} P_{vm_j} \tag{6.9}$$

$$\forall\, i \in 1..N \left(\sum_{j=1}^{K} (R_{vm_j} \cdot \neg vm_j^{hw_i}) + R_{hw_i} \cdot hw_i \geq \sum_{j=1}^{K} R_{vm_j} \right) \tag{6.10}$$

$$\forall\, i \in 1..N \left(\sum_{j=1}^{K} (P_{vm_j} \cdot \neg vm_j^{hw_i}) + P_{hw_i} \cdot hw_i \geq \sum_{j=1}^{K} P_{vm_j} \right) \tag{6.11}$$

$$\forall\, j \in 1..K \left(\sum_{i=1}^{N} vm_j^{hw_i} \geq 1 \right) \tag{6.12}$$

$$\forall\, j, i, k \in 1..K, 1..N, i + 1..K\ (\neg vm_j^{hw_i} + \neg vm_k^{hw_i}) \geq 1 \tag{6.13}$$

A função objetivo é a somatória dos *hardware* LIGADOS. As restrições (6.8) e (6.9) garantem que a somatória dos *hardware* LIGADOS conseguem acomodar todos os recursos das virtuais. As desigualdades (6.10) e (6.11) são os limites superiores do total de recurdos que cada *hardware* pode prover em relação às máquinas virtuais que poderão rodar neste equipamento. A restrição (6.12) define que cada máquina virtual deve estar alocada em algum *hardware*. Por fim a restrição (6.13) garante que a máquina virtual foi alocada em

exatamente um único *hardware*.

6.2.2 Discussão do PBFVMC

A formulação PBFVMC é uma melhoria em relação à formulação inicial, especialmente na quantidade de variáveis que é cortada pela metade. Esse corte é um facilitador ao resolvedor, já que o espaço de busca foi reduzido e também foram removidas as restrições não-lineares.

As restrições não-lineares, como as em (6.6) são difíceis de serem resolvidas e, além disso, a maioria, isso se não todos, os resolvedores traduzem as restrições não-lineares em uma instância equivalente linear. Esta transformação é facilmente feita, como avaliado em [57], porém muitos métodos introduzem uma quantidade significativa de variáveis auxiliares, aumentando o espaço de busca e causando um impacto direto no tempo de execução do resolvedor, especialmente quando as fórmulas possuem uma grande quantidade de *hardware* e virtuais.

As restrições (6.10) e (6.11) são apenas uma manipulação algébrica das restrições (6.4) e (6.11), retrabalhas de forma a deixar as variáveis hw_i no lado esquerdo da restrição para ditar que o hw_i deve estar ligado se alguma das $vm_j^{hw_i}$ estiverem alocadas neste *hardware*, após isso essas restrições estão normalizadas de forma que todos os coeficientes são não negativos e o operador relacional é o $\geq$.

As restrições (6.12) e (6.13) representam a restrição (6.6) que se baseiam no formato da formulação CNF do príncipio da casa dos pombos ao invés da formulação do Bin Packing. Enquanto a formulação *Bin Packing* é escrita estritamente na soma da restrição de igualdade que deve ser um como está a restrição (6.6), a formulação da casa dos pombos, por outro lado, é facilmente feita com cláusulas chegando a $(n + 1)$ cláusulas, onde n é o número de casas. As cláusulas ditam que os pombos devem ser colocados em alguma casa, como é a restrição (6.12), e então outras cláusulas devem gerar o conjunto que certificam que apenas um único pombo está dentro de cada casa, e isso é definido na restrição (6.13). O problema de utilizar esta formulação é que o número de cláusulas cresce rapidamente conforme o número de pombos cresce. Isso se torna mais crítico com

o problema de consolidação pois o número de virtuais e *hardware* podem chegar na casa dos milhares. Esta notação se torna difícil de ser utilizada já que o número de restrições cresce bastante gerando fórmula de dezenas de gigabytes.

O *pBCR* foi aplicado nesta nova formulação, a fim de identificar os efeitos em uma formulação muito próxima à formulação do princípio da casa dos pombos. Executamos em fórmulas pequenas para ficar com um tempo de execução reduzido e, ainda, poder entender as modificações feitas na fórmula, a fim de agregar na representação.

Após a aplicação do método *pBCR*, identificamos que a mudança ocorrida foi a de reescrever a restrição (6.13), que foi substituída pela restrição (6.14). Com esta nova restrição passamos a ter apenas uma restrição por máquina virtual para garantir a sua execução em um único *hardware* ao invés da combinação pareada, como acontecia na restrição (6.13).

$$\forall\, j \in 1..K \left(\sum_{i=1}^{N} \neg vm_j^{hw_i} \geq N - 1 \right) \tag{6.14}$$

A substituição da restrição (6.13) pela (6.14) faz com que a formulação *PBFVMC* tenha $(2 + 2 \times N + 2 \times K)$ restrições e $(N + N \times K)$ variáveis, ficando com apenas K restrições adicionais em relação à primeira formulação baseada no *Bin Packing* e com metade das variáveis.

Com esta nova formulação podemos gerar fórmulas com maiores quantidades de virtuais e *hardware* sem a preocupação com a quantidade exagerada de restrições que são geradas no modelo que utiliza a restrição (6.13), fazendo com que o arquivo com a fórmula ocupasse dezenas de *gigabytes* com apenas 128 *hardware* e 700 virtuais.

6.3 Avaliação Experimental

Para a implementação e avaliação das fórmulas pseudo-Booleanas foi escrito um programa, que faz a leitura do conjunto de *hardware* seguido pela quantidade de RAM e CPU disponíveis em cada *hardware*. Então é lido o conjunto de máquinas virtuais com seus requisitos de memória e processamento, por fim a fórmula gerada foi resolvida utilizando

os seguintes resolvedores: *Sat4j-PB* [42], *SCIP* [1] e *BSOLO* [45].

Foram geradas fórmulas para a formulação que se baseia na concepção do Bin Packing e a formulação *PBFVMC* já utilizando a restrição (6.14), que foi um conhecimento obtido com a utilização do *pBCR*.

Os experimentos foram executados em um Intel Xeon 2.1GHz com 256GB de memória e as fórmulas foram geradas a partir de dados do *Google Cluster Data* [1].

As fórmulas do experimento foram geradas com um crescimento progressivo dos conjuntos de trabalhos para identificar como as fórmulas e os tempos de otimização e decisão se comportavam. Um subconjunto de trabalho é o maior subconjunto de máquinas virtuais em que a soma dos recursos requeridos, de RAM e CPU, não ultrapassem uma porcentagem σ da soma dos recursos disponíveis de memória e processamento dos *hardware* disponíveis. No experimento assumimos que σ adota os seguintes valores: 25%, 50%, 75%, 85%, 90%, 95%, 98%, and 99%. Embora os experimentos tenham sido executados em diversos resolvedores, apenas os resultados do SAT4J-PB são mostrados, isso por ter sido o resolvedor com a melhor performance dentre os testados.

Devido a constantes de tempos curtas, selecionamos um conjunto de cinco subconjuntos de *hardware*. Os tamanhos de cada subconjunto são de **32**, **64**, **128**, **256**, **512** *hardware*. Para cada subconjunto de *hardware* foram utilizados os subconjuntos de trabalhos, definidos acima. A Tabela 6.1 mostra a quantidade de *hardware*, máquinas virtuais e o tamanho das fórmulas da primeira formulação e da PBFVMC.

Como resultado, a tabela 6.2 mostra os tempos obtidos para o conjunto de fórmulas. Para cada instância foi definido um tempo limite de 14400 segundos. Quando o resolvedor extrapola esse tempo e não encontra nenhuma solução é mostrado o texto TLE, de *Time Limit Exceeded*, fórmulas que não foram resolvidas em nenhumas das formulações são omitidas. Quando o resolvedor provou o resultado ótimo o tempo é escrito em negrito com o valor ótimo da função objetivo. Quando o resolvedor consegue encontrar uma valoração satisfatível mas não consegue provar a otimalidade do resultado, então, o tempo é grafado sem negrito e é mostrado o melhor resultado encontrado. A Tabela 6.3 mostra o

[1] http://code.google.com/p/googleclusterdata/

tempo gasto pelo resolvedor para decidir a fórmula, ou seja, para encontrar uma valoração satisfatível sem considerar a função objetivo.

6.4 Considerações Finais

A consolidação de máquinas virtuais é um tema bastante atual e, ainda sem uma solução definitiva. Os artigos publicados colocam pela primeira vez uma formulação do problema em restrições pseudo-Booleanas.

A última publicação, feita no BRACIS 2013, mostra que a nova formulação melhorou sensivelmente o tempo de decisão das fórmulas ficando demorada a otimização das fórmulas. Mostrando diretamente o impacto das técnicas utilizadas pelo *pBCR*, em um domínio do mundo real. Dentre as técnicas utilizadas temos que considerar em especial o fortalecimento, revisitado por Dixon [20], que causa o mesmo efeito que na formulação em CNF do princípio da casa dos pombos, como observado em usa tese.

Para este domínio é fácil identificar as implicações feitas pelo *pBCR* e com isso é possível gerar as fórmulas com as modificações que o *pBCR* faz em cima da representação original, evitando que seja necessário a execução de um pré-processador sempre que uma instância for gerada, economizando o tempo global de processamento da fórmula, uma vez que não se faz mais necessário o tempo de pré-processamento.

Tabela 6.1: Comparação dos tamanhos das fórmulas da primeira formulação e da PBFVMC. A tabela mostra os trabalhos de 25%, 50%, 75%, 85%, 90%, 95%, 98% and 99% para o subconjunto de 32, 64, 128, 256 e 512 *hardware*.

		Previous		PBFVMC	
HW	VMS	Vars	Constr	Vars	Constr
hw32-vm25p	98	6336	164	3168	262
hw32-vm50p	173	11136	239	5568	412
hw32-vm75p	278	17856	344	8928	622
hw32-vm85p	320	20544	386	10272	706
hw32-vm90p	325	20864	391	10432	716
hw32-vm95p	348	22336	414	11168	762
hw32-vm98p	364	23360	430	11680	794
hw32-vm99p	366	23488	432	11744	798
hw64-vm25p	174	22400	304	11200	478
hw64-vm50p	371	47616	501	23808	872
hw64-vm75p	559	71680	689	35840	1248
hw64-vm85p	629	80640	759	40320	1388
hw64-vm90p	665	85248	795	42624	1460
hw64-vm95p	707	90624	837	45312	1544
hw64-vm98p	712	91264	842	45632	1554
hw64-vm99p	713	91392	843	45696	1556

		Previous		PBFVMC	
HW	VMS	Vars	Constr	Vars	Constr
hw128-vm25p	368	94464	626	47232	994
hw128-vm50p	713	182784	971	91392	1684
hw128-vm75p	1048	268544	1306	134272	2354
hw128-vm85p	1155	295936	1413	147968	2568
hw128-vm90p	1277	327168	1535	163584	2812
hw128-vm95p	1321	338432	1579	169216	2900
hw128-vm98p	1368	350464	1626	175232	2994
hw128-vm99p	1410	361216	1668	180608	3078
hw256-vm25p	712	365056	1226	182528	1938
hw256-vm50p	1407	720896	1921	360448	3328
hw256-vm75p	2119	1085440	2633	542720	4752
hw256-vm85p	2372	1214976	2886	607488	5258
hw256-vm90p	2480	1270272	2994	635136	5474
hw256-vm95p	2583	1323008	3097	661504	5680
hw256-vm98p	2619	1341440	3133	670720	5752
hw256-vm99p	2678	1371648	3192	685824	5870

		Previous		PBFVMC	
HW	VMS	Vars	Constr	Vars	Constr
hw512-vm25p	1432	1467392	2458	733696	3890
hw512-vm50p	2771	2838528	3797	1419264	6568
hw512-vm75p	4035	4132864	5061	2066432	9096
hw512-vm85p	4431	4538368	5457	2269184	9888
hw512-vm90p	4745	4859904	5771	2429952	10516
hw512-vm95p	5068	5190656	6094	2595328	11162
hw512-vm98p	5319	5447680	6345	2723840	11664
hw512-vm99p	5402	5532672	6428	2766336	11830

Tabela 6.2: Tempos de execução por instância utilizando o resolvedor SAT4J-PB para a primeira formulação e PBFVMC. O tempo limite foi definido em 14400s e TLE representa que o tempo limite foi alcançado.

Formula	Previous	PBFVMC
hw32-vm25p	**249,897/7**	**191,912/7**
hw32-vm50p	35,696/16	4,134/16
hw32-vm75p	23,628/25	**772,657/24**
hw32-vm85p	1175,103/29	159,86/28
hw32-vm90p	108,361/31	**948,924/29**
hw32-vm95p	3442,92/32	**319,041/31**
hw32-vm98p	TLE	**45,651/32**
hw32-vm99p	TLE	**5566,491/32**
hw64-vm25p	4248,893/17	8,541/16
hw64-vm50p	6477,271/33	200,261/33
hw64-vm75p	8784,933/50	8608,38/47
hw64-vm85p	603,393/59	490,656/55
hw64-vm90p	1272,89/62	869,421/58
hw64-vm95p	TLE	679,719/62
hw64-vm98p	TLE	4135,757/64
hw64-vm99p	TLE	**240,642/64**
hw128-vm25p	TLE	10319,859/29
hw128-vm50p	14661,134/75	4856,869/64
hw128-vm75p	16209,656/105	12538,628/98
hw128-vm85p	11203,456/122	1117,772/115
hw128-vm90p	13491,676/128	11295,761/117
hw128-vm95p	TLE	65,916/128
hw256-vm25p	TLE	12381,653/68
hw256-vm50p	TLE	3576,626/136
hw256-vm75p	TLE	11468,942/204
hw256-vm85p	TLE	10537,747/230
hw256-vm90p	TLE	2704,592/243
hw256-vm95p	TLE	2003,068/255
hw256-vm98p	TLE	7737,502/256
hw512-vm25p	TLE	4471,005/140
hw512-vm50p	TLE	5406,047/281
hw512-vm75p	TLE	4378,66/408
hw512-vm85p	TLE	4919,328/461
hw512-vm90p	TLE	14426,6/487
hw512-vm95p	TLE	6864,151/510

Tabela 6.3: Tempo de execução com o resolvedor SAT4J-PB para encontrar alguma valoração satisfatível para a fórmula.

Formula	Previous	PBFVMC
hw32-vm25p	92,756	0,433
hw32-vm50p	35,643	0,542
hw32-vm75p	3,43	0,588
hw32-vm85p	4,516	0,911
hw32-vm90p	6,795	9,716
hw32-vm95p	3442,92	8,129
hw32-vm98p	TLE	45,589
hw32-vm99p	TLE	5566,28
hw64-vm25p	3118,029	0,706
hw64-vm50p	18,306	0,892
hw64-vm75p	50,687	1,15
hw64-vm85p	60,38	1,365
hw64-vm90p	121,006	1,423
hw64-vm95p	TLE	7,512
hw64-vm98p	TLE	4135,757
hw64-vm99p	TLE	240,538
hw128-vm25p	TLE	1,731
hw128-vm50p	4015,592	2,753
hw128-vm75p	5975,386	4,026
hw128-vm85p	7676,653	7,984
hw128-vm90p	13491,676	7,904
hw128-vm95p	TLE	65,916
hw256-vm25p	TLE	4,379
hw256-vm50p	TLE	14,244
hw256-vm75p	TLE	33,259
hw256-vm85p	TLE	48,298
hw256-vm90p	TLE	126,506
hw256-vm95p	TLE	389,329
hw256-vm98p	TLE	7737,502
hw512-vm25p	TLE	28,436
hw512-vm50p	TLE	162,289
hw512-vm75p	TLE	508,322
hw512-vm85p	TLE	287,437
hw512-vm90p	TLE	5604,022
hw512-vm95p	TLE	4222,892

CAPÍTULO 7

CONCLUSÃO

Muitos problemas do mundo real podem ser modelados utilizando restrições pseudo-Booleanas. Dado os avanços na tecnologia de SAT e de programação linear, é crucial que existam resolvedores pseudo-Booleanos capazes de atingir o mesmo nível de maturidade considerando a proximidade dos dois formalismos.

Nesta tese, antes de resolver os problemas pseudo-Booleanos, exploramos técnicas de pré-processamento de fórmulas, que são utilizadas há décadas na comunidade de programação linear. A comunidade de SAT tornou-se bastante importante, não somente pelos avanços dos resolvedores, mas também pelo crescente aumento de técnicas de pré-processamento de fórmulas, que até então haviam sido pouco tratadas no cenário pseudo-Booleano.

Como resultado, temos o *pBCR* um algoritmo capaz de tratar diversas etapas de pré-processamento nas fórmulas. Os resultados experimentais mostraram que, de fato, existe um impacto no tempo dos resolvedores nas fórmulas. Em diversos casos obtivemos um impacto positivo, ou seja, o tempo de resolução da fórmula foi menor que sem o pré-processamento, e em outros casos obtivemos um impacto negativo.

O impacto de pré-processamento, em fórmulas pseudo-Booleanas, foi considerado baixo em trabalhos recentes, como o *Preprocessing in Pseudo-Boolean Optimization: An Experimental Evaluation* [50], em 2009. Neste trabalho foi feita uma releitura das técnicas de pré-processamento em SAT, aplicando-as em fórmulas pseudo-Booleanas, as quais obtiveram um certo impacto positivo, mas não tão grande quanto nas fórmulas em CNF. Com o *pBCR* pudemos fazer uma análise mais ampla, aplicando as técnicas vastamente utilizadas pela comunidade de pesquisa operacional, identificando ganhos em desempenho.

A primeira contribuição técnica desta tese foi mostrar que é possível trazer técnicas de pré-processamento da comunidade de programação inteira mista para os problemas

pseudo-Booleanos e SAT e, ainda sim, alterar o tempo que os resolvedores utilizam em cada instância modificada. Com o *pBCR* pudemos identificar como cada uma das técnicas altera o comportamento do resolvedor perante as fórmulas utilizadas. Diversas fórmulas possuíram um desempenho muito aquém do esperado, mas tivemos instâncias que só puderam ser resolvidas dentro do limite de tempo empregado, com as configurações de pré-processamento utilizadas. Outro fator importante foi a identificação do impacto da aridade das fórmulas nos resolvedores que, mesmo com as técnicas de propagação rápida e literais vigiados, sofreram um impacto de tempo, tendo as fórmulas com as restrições com mais literais.

O impacto positivo mais surpreendente foi que a aplicação da sondagem nas fórmulas de otimização pseudo-Booleana ajudaram o resolvedor SCIP a encontrar a solução ótima, nessas fórmulas, mais rapidamente. Embora o SCIP já faça a utilização da sondagem, uma executada anteriormente ao pré-processamento do SCIP gerou uma fórmula mais enxuta e que auxiliou o resolvedor a explorar outros ramos da fórmulas.

Os resolvedores LINGELING, GLUCOSE e CLASP também fazem algum pré-processamento e, mesmo assim, possuem seu tempo de processamento alterado quando se utiliza as fórmulas pré-processadas pelo *pBCR*. O interessante deste pré-processamento é o fato de ser feito sem o controle do resolvedor, evitando que este guarde informações sobre a fórmula antes dela ser pré-processada. Este aspecto pode causar uma perda de desempenho pelo fato de algumas informações estruturais desaparecerem da fórmula, mas em outros casos, uma fórmula em uma representação mais forte, pode auxiliar o resolvedor a não perder tempo em variáveis e restrições que afetam pouco no desencadeamento das decisões. Isto se mostrou muito importante no experimento de otimização com o resolvedor SCIP.

As conjecturas formuladas neste tese, trazem um questionamento sobre a generalização dos conceitos de aprendizado e subjugação, sendo um resultado importante no conceito de pré-processamento, uma vez que processamentos que demandam muito tempo de processamento possam ser simplificados por suas generalizações causando um impacto equivalente para o resolvedor, mas uma demanda de tempo menor para efetuar o pré-processamento.

A conjectura 1 permite generalizar o conceito de aprendizado por desigualdades geradas pelo clique no grafo, facilitando a inserção de restrições aprendidas sem a necessidade de uma busca por cliques maximais.

A conjectura 2 permite simplificar o processo de subjugação das restrições envolvidas no fortalecimento. Agora não é mais necessário avaliar se os modelos da restrição fortalecida é equivalente aos modelos das restrições envolvidas no processo, evitando que uma enumeração da tabela verdade seja feita desnecessariamente.

Também foi objeto de estudo desta tese, a aplicação de uma formulação pseudo-Booleana para um problema do mundo real. A vista disto, usamos a consolidação de máquinas virtuais como o estudo de caso. O desenvolvimento desta formulação foi feito em dois passos. Primeiramente foi criada uma formulação mais parecida com o problema da mochila, onde obtivemos um resultado muito ruim, mesmo para decisão nas instâncias geradas. O segundo passo foi uma formulação baseada no princípio da casa dos pombos, porém utilizando uma casa onde possam entrar mais pombos, desde que todos eles não extrapolem o limite da casa. A segunda formulação mostrou ser muito mais compatível com as tecnologias dos resolvedores, pois foram resolvidas mais rapidamente e, quando se utiliza a otimização, mais problemas puderam ser provados.

Além da criação da formulação do problema, as fórmulas geradas pelos artigos foram enviadas à competição de resolvedores pseudo-Booleanos de 2015, integrando os domínios utilizados para classificar os resolvedores.

Trabalhos Futuros

Explorar mais a fundo outros parâmetros para a sondagem. Além de uma porcentagem das variáveis, devemos experimentar conjuntos de variáveis seguindo heurísticas. Já que o pré-processamento pode gerar um grande impacto no resolvedor devemos identificar variáveis mais relevantes para se efetuar a sondagem. É provável que após efetuar a sondagem em algumas variáveis seja possível identificar as variáveis que mais se relacionam com o processo de busca, gerando um processo de escolha de variáveis baseado na heurística VSIDS, utilizada pelos resolvedores durante o processo de busca.

Explorar os mecanismos de pré-processamento em CNF e trazer parte de suas contribuição para pseudo-Boolean, assim como fizemos com a programação linear. Conforme já identificado por Ruben Martin [50], várias técnicas podem ser aplicadas em restrições pseudo-Booleanas e por isso uma atenção especial a essas técnicas deve ser observada e identificar quais poderiam ser adicionadas ao *pBCR*.

Aprofundar e provar as conjecturas apresentadas. Tivemos algumas conjecturas nesta tese que dem ser, posteriormente, provadas já que as evidências empíricas sugerem que elas devem ser verdade. Em especial a conjectura 2 que trata da subjugação de restrições. A conjectura 1 que trata do aprendizado avançado deve, também, ser implementada no *pBCR* para identificar se suas restrições afetam de forma positiva o tempo do resolvedor.

Estender a formulação PBFVMC, adicionando as restrições de afinidade e outras que a comunidade de computação em nuvem julgam importantes. Já que este conjunto de fórmulas tratam de um problema do mundo real, e se tornam muito interessantes para as competições, podemos agregar novas restrições para poder representar melhor as demandas da comunidade de Computação em Nuvem, gerando fórmulas mais próximas do ambiente real.

BIBLIOGRAFIA

[1] Tobias Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, e M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Relatório técnico, EECS Department, University of California, Berkeley, 2009.

[3] Gilles Audemard e Laurent Simon. Predicting learnt clauses quality in modern sat solvers. *IJCAI*, volume 9, páginas 399–404, 2009.

[4] Peter Barth e Im Stadtwald. A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization, 1995.

[5] Anton Belov, Antonio Morgado, e Joao Marques-Silva. Sat-based preprocessing for maxsat (extended version). *arXiv preprint arXiv:1310.2298*, 2013.

[6] Armin Biere. Lingeling essentials, a tutorial on design and implementation aspects of the the sat solver lingeling. Daniel Le Berre, editor, *POS-14*, volume 27 of *EPiC Series*, páginas 88–88. EasyChair, 2014.

[7] Armin Biere, Marijn J. H. Heule, Hans van Maaren, e Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February de 2009.

[8] Robert E Bixby e Donald K Wagner. A note on detecting simple redundancies in linear systems. *Operations research letters*, 6(1):15–17, 1987.

[9] H. Bohm. Report on a SAT competition. Relatório técnico, Technical report, none.

[10] Endre Boros e Peter L Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1):155–225, 2002.

[11] R. Bossche, K. Vanmechelen, e J. Broeckhove. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. *Proccedings of IEEE 3rd International Conference on Cloud Computing*, páginas 228 –235. IEEE Computer Society, julho de 2010.

[12] Vasek Chvatal. *Linear programming*. Macmillan, 1983.

[13] Stephen A. Cook. The complexity of theorem-proving procedures. *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, páginas 151–158, New York, NY, USA, 1971. ACM.

[14] A. Corradi, M. Fanelli, e L. Foschini. Increasing Cloud Power Efficiency through Consolidation techniques. *Proceeding of IEEE Symposium on Computers and Communications.*, páginas 129–134. IEEE Computer Society, junho de 2011.

[15] Harlan Crowder, Ellis L Johnson, e Manfred Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[16] M. Davis, G. Logemann, e D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):397, 1962.

[17] M. Davis e H. Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[18] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[19] BL Dietrich e LF Escudero. Coefficient reduction for knapsack-like constraints in 0–1 programs with variable upper bounds. *Operations Research Letters*, 9(1):9–14, 1990.

[20] Heidi Dixon. *Automating pseudo-boolean inference within a dpll framework*. Tese de Doutorado, University of Oregon, 2004.

[21] Heidi E Dixon e Matthew L Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. *AAAI/IAAI*, páginas 635–640, 2002.

[22] Heidi E. Dixon, Matthew L. Ginsberg, David K. Hofer, Eugene M. Luks, e Andrew J. Parkes. Generalizing boolean satisfiability iii: Implementation. *J. Artif. Intell. Res. (JAIR)*, 23:441–531, 2005.

[23] Heidi E. Dixon, Matthew L. Ginsberg, Eugene M. Luks, e Andrew J. Parkes. Generalizing boolean satisfiability ii: Theory. *J. Artif. Intell. Res. (JAIR)*, 22:481–534, 2004.

[24] Heidi E. Dixon, Matthew L. Ginsberg, e Andrew J. Parkes. Generalizing boolean satisfiability i: Background and survey of existing work. *J. Artif. Intell. Res. (JAIR)*, 21:193–243, 2004.

[25] Niklas Eén e Armin Biere. Effective preprocessing in sat through variable and clause elimination. *Theory and Applications of Satisfiability Testing*, páginas 61–75. Springer, 2005.

[26] Niklas Eén e Niklas Sörensson. An extensible sat-solver. Enrico Giunchiglia e Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, páginas 333–336. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-24605-3_37.

[27] T. C. Ferreto, M. a.S. Netto, R. N. Calheiros, e C. a.F. De Rose. Server Consolidation with Migration Control for Virtualized Data Centers. *Journal of Future Generation Computer Systems*, 27(8):1027–1034, outubro de 2011.

[28] Jon William Freeman. *Improvements to propositional satisfiability search algorithms*. Tese de Doutorado, Philadelphia, PA, USA, 1995. UMI Order No. GAX95-32175.

[29] M. Gebser, B. Kaufmann, e T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89, 2012.

[30] Monique Guignard e Kurt Spielberg. Logical reduction methods in zero-one programming—minimal preferred variables. *Operations Research*, 29(1):49–74, 1981.

[31] Peter L Hammer e Sergiu Rudeanu. *Boolean methods in operations research and related areas*, volume 7. Springer Science & Business Media, 1968.

[32] R.R. Harmon e N. Auseklis. Sustainable IT Services: Assessing the Impact of Green Computing Practices. *Proceedings of International Conference on Management of Engineering & Technology.*, páginas 1707–1717. IEEE Computer Society, agosto de 2009.

[33] Karla L Hoffman e Manfred Padberg. Improving lp-representations of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing*, 3(2):121–134, 1991.

[34] J. Huang. The effect of restarts on the efficiency of clause learning. *Proceedings of the 20th international joint conference on Artifical intelligence*, páginas 2318–2323. Morgan Kaufmann Publishers Inc., 2007.

[35] Ronald L. Iman e James M. Davenport. Approximations of the critical region of the fbietkan statistic. *Communications in Statistics - Theory and Methods*, 9(6):571–595, 1980.

[36] Matti Järvisalo, Armin Biere, e Marijn Heule. Blocked clause elimination. *Tools and Algorithms for the Construction and Analysis of Systems*, páginas 129–144. Springer, 2010.

[37] Robert G. Jeroslow e Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990. 10.1007/BF01531077.

[38] Ellis L Johnson e Manfred W Padberg. Degree-two inequalities, clique facets, and biperfect graphs. *North-Holland Mathematics Studies*, 66:169–187, 1982.

[39] Henry A. Kautz, Bart Selman, e Jörg Hoffmann. SatPlan: Planning as satisfiability. *Abstracts of the 5th International Planning Competition*, 2006.

[40] A Klar. Cutting planes in mixed integer programming. Dissertação de Mestrado, Technische Universitat Berlin, Alemanha, 2006.

[41] Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96:149–176, 1999.

[42] D. Le Berre. SAT4j: a Reasoning Engine in Java based on the SATisfiability Problem. `http://www.sat4j.org`.

[43] N. Leavitt. Is Cloud Computing Really Ready for Prime Time? *Journal of Computer*, 42(1):15–20, janeiro de 2009.

[44] C. M. Li e F. Manyà. *MaxSAT, Hard and Soft Constraints*, capítulo 19, páginas 613–631. Volume 185 of Biere et al. [7], February de 2009.

[45] V. Manquinho. BSOLO: A Solver for Pseudo-Boolean Constraints. `http://sat.inesc-id.pt/~vmm/research/`.

[46] Vasco M. Manquinho, João P. Marques Silva, e Jordi Planes. Algorithms for weighted boolean optimization. Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, páginas 495–508. Springer, 2009.

[47] J. Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. *Progress in Artificial Intelligence*, páginas 850–850, 1999.

[48] J. P. Marques-Silva, I. Lynce, e S. Malik. *Conflict-Driven Clause Learning SAT Solvers*, capítulo 4, páginas 131–153. Volume 185 of Biere et al. [7], February de 2009.

[49] J.P. Marques-Silva e K.A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *Computers, IEEE Transactions on*, 48(5):506–521, 2002.

[50] Ruben Martins, Inês Lynce, e Vasco Manquinho. Preprocessing in pseudo-boolean optimization: An experimental evaluation. *Constraint Modelling and Reformulation (ModRef'09)*, páginas 87, 2009.

[51] M. Marzolla, O. Babaoglu, e F. Panzieri. Server Consolidation in Clouds through Gossiping. *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, páginas 1–6. IEEE Computer Society, junho de 2011.

[52] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, e S. Malik. Chaff: Engineering an efficient SAT solver. *Design Automation Conference, 2001. Proceedings*, páginas 530–535. IEEE, 2005.

[53] P. Nemenyi. *Distribution-free Multiple Comparisons*. Princeton University, 1963.

[54] Bruno César Ribas. Satisfatibilidade não-clausal restrita às variáveis de entrada. Dissertação de Mestrado, Universidade Federal do Paraná, 2011.

[55] Bruno Cesar Ribas, Rubens Massayuki Suguimoto, Razer A.N.R. Monta no, Fabiano Silva, e Marcos A. Castilho. Pbfvmc: A new pseudo-boolean formulation to virtual-machine consolidation. Aurora Pozo, editor, *BRACIS 2013*, volume 7637 of *IEEE*, páginas 361–370. IEEE, 2013.

[56] Bruno Cesar Ribas, Rubens Massayuki Suguimoto, Razer A.N.R. Monta no, Fabiano Silva, Luis de Bona, e Marcos A. Castilho. On modelling virtual machine consolidation to pseudo-boolean constraints. Juan Pavón, Néstor D. Duque-Méndez, e Rubén Fuentes-Fernández, editors, *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637 of *Lecture Notes in Computer Science*, páginas 361–370. Springer, 2012.

[57] O. Roussel e V. Manquinho. *Pseudo-Boolean and Cardinality Constraints*, capítulo 22, páginas 695–733. Volume 185 of Biere et al. [7], February de 2009.

[58] MW Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–445, 1994.

[59] B. Selman, H. Kautz, e B. Cohen. Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–532, 1996.

[60] B. Selman, H. Levesque, e D. Mitchell. A new method for solving hard satisfiability problems. páginas 440–446, 1992.

[61] Carsten Sinz et al. Sat race 2008, 2012.

[62] Zbigniew Stachniak e Anton Belov. Speeding-up non-clausal local search for propositional satisfiability with clause learning. *Proceedings of the 11th international conference on Theory and applications of satisfiability testing*, SAT'08, páginas 257–270, Berlin, Heidelberg, 2008. Springer-Verlag.

[63] N. Sörensson e N. Eén. MiniSat 2.1 and MiniSat++ 1.0—SAT Race 2008 Editions. *SAT 2009 competitive events booklet: preliminary version*, páginas 31, 2009.

[64] G.S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in constructive mathematics and mathematical logic*, 2(115-125):10–13, 1968.

[65] L. M. Vaquero, L. Rodero-Merino, J. Caceres, e M. Lindner. A Break in the Clouds: towards a Cloud Definition. *Journal of ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[66] Miroslav N. Velev e Randal E. Bryant. Formal verification of superscalar microprocessors with multicycle functional units. páginas 112–117, 2000.

[67] W. Vogels. Beyond Server Consolidation. *Journal of ACM Queue*, 6(1):20, janeiro de 2008.

[68] Laurence A Wolsey e George L Nemhauser. *Integer and combinatorial optimization.* John Wiley & Sons, 1988.

[69] Lin Xu, Frank Hutter, Holger H Hoos, e Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, páginas 565–606, 2008.

[70] Ramin Zabih. A rearrangement search strategy for determining propositional satisfiability. *in Proceedings of the National Conference on Artificial Intelligence*, páginas 155–160, 1988.

[71] Hantao Zhang. Sato: An efficient prepositional prover. William McCune, editor, *Automated Deduction—CADE-14*, volume 1249 of *Lecture Notes in Computer Science*, páginas 272–275. Springer Berlin / Heidelberg, 1997. 10.1007/3-540-63104-6_28.

APÊNDICE A

ANÁLISE DE SIGNIFICÂNCIA ESTATÍSTICA

A análise de significância estatística é bastante utilizada em diversas áreas da computação, mas não na avaliação experimental da comunidade de SAT. Por isso introduzimos aqui os conceitos para a avaliação de Friedman e seus pós-testes para permitir melhor entendimento de algumas avaliações efetuadas.

A aplicação do teste de Friedman [18] é recomendado para analisar a existência de diferença estatisticamente significativa entre os desempenhos de K modelos sobre N conjunto de dados. Este teste é uma versão não-paramétrica equivalente ao ANOVA medidas repetidas, para análise de dados pareados. Esse teste baseia-se na comparação de posições de desempenhos (*rank*). Para cada conjunto de dados, cada um dos modelos avaliados são associados a uma posição, de modo ordenado de acordo com seus desempenhos, dos melhores para os piores. Em caso de empates, valores médios de posição são atribuídos. Após, é determinado o *rank* médio para cada um dos K modelos avaliados. Assim, seja R_{ij} a posição de desempenho do modelo j, $(1 \leq j \leq K)$, para um conjunto de dados i, $(1 \leq i \leq N)$, o *rank* médio R_j é calculado conforme a equação A.1.

$$R_j = \frac{\sum_{i=1}^{N} R_{ij}}{N} \tag{A.1}$$

Sob a hipótese nula de que o desempenho de todos os modelos comparados são equivalentes, e que portanto seus valores de *rank* médio são iguais, a estatística de Friedman é expressa pela equação A.2.

$$\chi_F^2 = \frac{12N}{K(K+1)} \left[\sum_{j=1}^{K} R_j^2 - \frac{K(K+1)^2}{4} \right] \tag{A.2}$$

No entanto, é recomendada a utilização de uma versão menos conservadora do teste de Friedman, conforme apontado em [18]. Esta versão foi proposta em [35], a qual é distribuída de acordo com a distribuição F com $(K-1)$ e $(K-1)(N-1)$ graus de liberdade. O cálculo dessa estatística é apresentado na equação A.3.

$$F_F = \frac{(N-1)\chi_F^2}{N(K-1)\chi_F^2} \tag{A.3}$$

Quando a hipótese nula é rejeitada pelo teste de Friedman, isso implica a existência de diferença significativa de desempenhos, no entanto não permite identificar quais modelos apresentam diferença. Nesse cenário, o pós-teste de Nemenyi [53] pode ser utilizado para detectar quais diferenças entre os modelos são significativas [18]. De acordo com esse teste, a eficácia de dois métodos é significativamente diferente sempre que seus correspondentes *ranks* médios diferem por pelo menos um determinado valor de diferença crítica (CD), expresso pela equação A.4.

$$\text{CD} = q_\alpha \sqrt{\frac{K(K+1)}{6N}} \tag{A.4}$$

Na tabela A.1 são apresentados os valores de q_α com $\alpha = 0.05$, para diferentes quan-

tidades de métodos a serem comparados, utilizando o pós-teste de Nemenyi.

K	2	3	4	5	6	7	8	9	10
Nemenyi	1,960	2,343	2,569	2,728	2,850	2,949	3,031	3,102	3,164

Tabela A.1: Valores de $q_{0.05}$ para diferentes valores de K no pós-teste de Nemenyi.