Relatório do Problema de Satisfazibilidade Booleana

1 – Descrição da solução:

O programa se divide em três partes principais:

1. Leitura do arquivo e criação de lista com clausulas e variáveis.
2. Teste se a conjunto dos valores booleanos tornam a formula proposicional verdadeira.
3. Mudança no conjunto dos valores booleanos para novo teste.

A primeira parte ao receber o nome do arquivo, analisa linha a linha por meio de um "loop", observando o primeiro caractere da linha. Se esse caractere for "p", o programa já identifica o número de clausulas e variáveis. Após isso, lê linha a linha para identificar as cláusulas, verificando onde cada variável se encaixa. O programa utiliza as funções "strip" e "split" para separar a linha em blocos e poder organizar as informações. No final, se forma uma lista de listas (no mesmo formato do exemplo do enunciado do EP: "fórmula= [[1,-5,4], [-1,3,4], [2,-3,-4]]").

A segunda parte, a partir da lista gerada pela leitura dos arquivos, testa usando a lista utilizando uma "String" que armazena o valor de cada variável. Essa "String" guarda "0" ou "1" em cada posição; assim a posição [0] guarda o valor da variável X_1, a posição [1] guarda o valor da variável X e assim por diante.

O teste é feito por meio de um "loop" triplo: o " loop" principal é baseado num "while", ocorre enquanto o número de interações é inferior ao número de possibilidades de conjunto de valores booleanos (calculado com dois elevado ao número de variáveis) e enquanto todas clausulas ainda não foram testadas. O segundo "loop" age enquanto todas cláusulas não foram testadas e uma cláusula teve todos seus valores testados, mas não tenha iniciado o próximo "loop" - o terceiro "loop" traz a condição para iniciar o próximo "loop" do segundo. O terceiro "loop" roda três vezes no máximo; pois cada clausula tem três variáveis, assim a ideia é ocorrer um "break". Este acontece quando uma variável não possui negação e seu valor é 1 ou possui negação e seu valor é 0.

Dessa forma, o programa tenta não chegar no limite de interações - senão retorna que não há solução. E percorrer todas as cláusulas em algum conjunto de valores, assim sair do "loop" principal e poder retornar o valor das variáveis as quais fizeram a formula verdadeira.

A terceira parte possui dois algoritmos o "BruteForce" e "RandomSearch": o primeiro cria a primeira "string" para teste com todas variáveis 1 (verdadeiro). Depois transforma a "string" em um binário e subtrai um, e volta para uma string; adicionando zeros a frente, se precisar para manter o tamanho - pela perda de uma casa binaria. O segundo cria aletoriamente um conjunto de valores toda vez, usando como "seed" da função "random" o número de interações para evitar repetições.

2 -Análise do arquivo teste:

Ambos algoritmos resolveram o problema com uma interação. O "BruteForce" porque a primeira combinação testada é: X_1=1 X_2=1 X_3=1 X_4=1. E o "RandomSearch" como uma usa sempre o "seed" 1 na primeira vez, obteve sucesso com a combinação: X_1=1 X_2=1 X_3=0 X_4=1.

3 - Análise dos resultados:

Arquivo	uf20-01.cnf		uf20-02.cnf		uf20-03.cnf		uf20-04.cnf		uf20-05.cnf	
Algoritmo	Brute	Random	Brute	Random	Brute	Random	Brute	Random	Brute	Random
Interações	441111	121742	460430	103097	33123	855260	317800	252371	1006155	957144
Tempo(s)	21,64	26,79	22,9	17,17	1,52	150,63	17,43	43,33	43,46	197,53

O algoritmo "BruteForce" se mostrou muito rápido para montar novas combinações de valores, enquanto o "RandomSearch" foi lento – provavelmente pelas funções utilizadas do python.

Além disso, o "BruteFoce" mostrou certa constância nos problemas de 20 variáveis utilizando na casa de centenas de milhares de interações. Enquanto o "RandomSearch" em quase todos os arquivos utilizou menos interações, porém demorava mais porque gastava muito tempo na geração de novos valores.

Dessa forma, O "BruteForce" se mostrou constante e rápido, porém menos eficiente que "RandomSearch"; mesmo com a demora para criação de valores – o que poderia ser corrigido utilizando outra função ou linguagem-, e a inconsistência no número de interações de cada arquivo devido a aleatoriedade do método.

Nome: Alessandro Brugnera Siva NUSP:10334040 Turma: 01