UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

ALUNOS:
PEDRO HENRIQUE DE ALMEIDA E PADUA 537674
JOÃO VICTOR GOMES DOS SANTOS 536203

PROFESSOR:
PROFº DOUTOR ALEXANDRE MATOS ARRUDA

Relatório da implementação de um SATPLAN para resolução do problema do mundo dos blocos

Introdução

O mundo dos blocos se baseai em uma mesa com blocos que podem ser empilhados uns sobre os outros, e uma garra robótica que os movimenta. O objetivo é organizar os blocos empilhados em determinada ordem com o menor numero de passos possíveis e seguindo as regras para realizar as operações.

O objetivo do trabalho proposto é achar uma solução computacional para que seja realizado a ordenação dos blocos, baseado em pré-condições, pós-condições e ações permitidas utilizando a biblioteca da linguagem python pysat.

Fundamentação teórica

O problema SAT é uma verificão se uma fórmula pode ser verdadeira para algum valor verdadeiro ou falso das suas variáveis. Já o SATSOLVER é um algoritimo que visa resolver esse problema

Dado um conjunto de cláusulas na forma normal conjuntiva (FNC), o SATSOLVER busca um valor verdadeiro ou falso para cada variável da fórmula que faça com que todas as cláusulas sejam verdadeiras.

O problema do mundo dos blocos, é um problema clássico em teoria dos algoritmos e inteligência artificial.

Metodologia

Para implementação do SATSOLVER foi utilizado a linguagem de programação python e as bibliotecas pysat, sys, time, memory_profile e psUntil, assim como 2 classe fornecidas pelo professor, SatPlanInstance e SatPlanInstanceMapper.

A implementação foi realizada em uma só função Main, em que todas as operações são envolvidas por um laço de repetição for, que vai de 1 ate n(infinito), que representa os níveis em que mudanças serão feitas nos blocos, a cada passo buscando chegar mais perto do estado final. A cada nível são mapeadas as ações, pré-condições, pós condições, estado atual, e tudo isso é enviado para o SATSOLVER, no final é exibido no terminal o resultado final, assim como o tempo e memoria gastos pelo algorítimo.

Experimentos e Resultados

Para teste do algorítimo foram executadas 13 instancias, ás quais a que demandou maior tempo e memória foi a “blocks-17-0.strips” que demorou 02 horas 18 minutos 1 segundo, com utilização total de memoria de:

181.92 MB

divididas em:

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    28     30.8 MiB     30.8 MiB           1   @profile
    29                                         def main():
    30     31.4 MiB      0.6 MiB           1       satPlanInstance = SatPlanInstance(sys.argv[1])
    31     31.4 MiB      0.0 MiB           1       instanceMapper  = SatPlanInstanceMapper()
    32     31.4 MiB      0.0 MiB           1       instanceMapper.add_list_of_literals_to_mapping(satPlanInstance.get_atoms())
    33     31.5 MiB      0.0 MiB           1       gluc = Glucose3()
    34     31.5 MiB      0.0 MiB           1       model= []
    35
    36     31.5 MiB      0.0 MiB           1       n_level = sys.maxsize
    37     34.0 MiB    -34.3 MiB          17       for i in range(1, n_level):
    38     34.0 MiB    -33.3 MiB          17           a = create_literals_for_level_from_list(i, satPlanInstance.get_actions())
    39                                                 # print(a)
    40     34.0 MiB    -32.0 MiB          17           instanceMapper.add_list_of_literals_to_mapping(a)
    41     34.0 MiB    -32.0 MiB          17           actions_list = instanceMapper.get_list_of_literals_from_mapping(a)
    42                                                 # print(actions_list)
    43
    44     35.8 MiB -17872.9 MiB        9843           for actions in satPlanInstance.get_actions():
    45     35.8 MiB -17860.2 MiB        9826               pre_condition = create_literals_for_level_from_list(i, satPlanInstance.get_action_preconditions(actio
ns))
    46                                                     # print(pre_condition)
    47     35.8 MiB -17860.3 MiB        9826               acoes_level= create_literal_for_level(i, actions)
    48                                                     #ALTEREI AQUI
    49     35.8 MiB -17860.3 MiB        9826               instanceMapper.add_list_of_literals_to_mapping(pre_condition)
    50                                                     #instanceMapper.add_list_of_literals_to_mapping(pre_condition)
    51     35.8 MiB -61947.4 MiB       34102               for pre in pre_condition:
    52     35.8 MiB -44103.8 MiB       24276                   mapped_condition = instanceMapper.get_literal_from_mapping(pre)
    53     35.8 MiB -44085.6 MiB       24276                   gluc.add_clause([-instanceMapper.get_literal_from_mapping(acoes_level), mapped_condition])
    54
```

```
    54
    55     35.8 MiB -17843.2 MiB        9826              pos_condition = create_literals_for_level_from_list(i+1, satPlanInstance.get_action_posconditions(act
ions))
    56     35.8 MiB -17842.0 MiB        9826              instanceMapper.add_list_of_literals_to_mapping(pos_condition)
    57     35.8 MiB -105998.2 MiB       58378              for pos in pos_condition:
    58     35.8 MiB -88157.8 MiB        48552                 mapped_condition2 = instanceMapper.get_literal_from_mapping(pos)
    59     35.8 MiB -88153.9 MiB        48552                 gluc.add_clause([-instanceMapper.get_literal_from_mapping(acoes_level), mapped_condition2])
    60
    61
    62                                                  # print(f'Inicial: {satPlanInstance.get_initial_state()}')
    63     35.9 MiB    -30.0 MiB           17           initials_states = create_state_from_literals(satPlanInstance.get_initial_state(), satPlanInstance.get_sta
te_atoms())
    64     35.9 MiB    -30.2 MiB           17           estado_inicial = create_literals_for_level_from_list(1, initials_states)
    65     35.9 MiB    -30.3 MiB           17           instanceMapper.add_list_of_literals_to_mapping(estado_inicial)
    66     35.9 MiB  -9810.8 MiB         5525           for estado in instanceMapper.get_list_of_literals_from_mapping(estado_inicial):
    67     35.9 MiB  -9780.7 MiB         5508               gluc.add_clause([estado])
    68
    69                                                  # print(f'Final: {satPlanInstance.get_final_state()}')
    70     35.9 MiB    -30.2 MiB           17           estado_final = create_literals_for_level_from_list(i, satPlanInstance.get_final_state())
    71                                                  # print(estado_final)
    72     35.9 MiB    -30.2 MiB           17           instanceMapper.add_list_of_literals_to_mapping(estado_final)
    73
    74     35.9 MiB   -512.9 MiB          289           for estado in instanceMapper.get_list_of_literals_from_mapping(estado_final):
    75     35.9 MiB   -482.7 MiB          272               gluc.add_clause([estado])
    76
    77    159.2 MiB  -4835.5 MiB          153           for j in range(1,i):
    78    159.2 MiB -3058438.3 MiB        78744                 for acao in satPlanInstance.get_actions():
    79    159.1 MiB -3053620.7 MiB        78608                     poscondition = satPlanInstance.get_action_posconditions(acao)
    80    159.1 MiB -3053033.9 MiB        78608                     state= satPlanInstance.get_state_atoms()
    81    159.2 MiB -992293047.5 MiB    25547600                       for estado in state:
```

```
    81    159.2 MiB -992293047.5 MiB    25547600                       for estado in state:
    82    159.2 MiB -989241043.0 MiB    25468992                           if estado not in poscondition:
    83                                                                       # Mapeamento de Literais
    84    159.2 MiB -981700118.9 MiB    25274784                               level_estado = create_literal_for_level(j, estado)
    85    159.2 MiB -981700157.1 MiB    25274784                               level_acao = create_literal_for_level(j, acao)
    86    159.2 MiB -981700206.4 MiB    25274784                               level_prox_estado = create_literal_for_level(j + 1, estado)
    87
    88                                                                       # Adição de Cláusulas ao Solver
    89    159.2 MiB -981698868.4 MiB    25274784                               gluc.add_clause([instanceMapper.get_literal_from_mapping(level_estado), -instanceMapp
er.get_literal_from_mapping(level_acao), -instanceMapper.get_literal_from_mapping(level_prox_estado)])
    90    159.2 MiB      0.0 MiB          153           for j in range(1, i):
    91    159.2 MiB      0.4 MiB          136               level_actions = create_literals_for_level_from_list(j, satPlanInstance.get_actions())
    92    159.2 MiB      0.1 MiB          136               mapped_actions = instanceMapper.get_list_of_literals_from_mapping(level_actions)
    93    159.2 MiB      0.3 MiB          136               gluc.add_clause(mapped_actions)
    94
    95
    96    223.6 MiB    309.0 MiB           17           gluc.solve()
    97    223.7 MiB    -42.0 MiB           17           if gluc.get_model() != None:
    98    181.4 MiB    -42.3 MiB            1               print('resolvido')
    99    181.4 MiB      0.0 MiB            1               model = gluc.get_model()
   100    182.4 MiB      0.8 MiB        16561               for j in list(gluc.get_model()):
   101    182.4 MiB      0.0 MiB        16560                   if j > 0:
   102    182.4 MiB      0.2 MiB          175                       action = instanceMapper.get_literal_from_mapping_reverse(j)
   103    182.4 MiB      0.0 MiB          175                       print(instanceMapper.get_literal_from_mapping_reverse(j))
   104    182.4 MiB      0.0 MiB            1               break
   105                                                  else:
   106    223.8 MiB      0.1 MiB           16               print(f'{i}não resolvido')
   107     34.0 MiB  -1302.6 MiB           16               gluc.delete()
   108     34.0 MiB    -34.1 MiB           16               gluc = Glucose3()
```

E apresentou o seguinte resultado:

1_stack_h_c
1_handempty
1_clear_h
1_clear_q
1_clear_p
1_clear_l
1_clear_g
1_on_a_j
1_on_f_e
1_on_l_f
1_ontable_m
1_on_c_o
1_on_q_a
1_ontable_n
1_on_d_c
1_ontable_o
1_on_g_d
1_on_e_k
1_ontable_p
2_on_h_c
1_on_b_m
1_on_h_n
1_on_j_i
1_on_i_b
1_ontable_k
2_stack_f_d
3_on_f_d
3_on_h_c

3_stack_b_i
4_on_f_d
4_on_b_i
4_on_h_c
4_stack_a_g
5_on_a_g
5_on_f_d
5_on_b_i
5_on_h_c
5_stack_g_b
6_on_a_g
6_on_f_d
6_on_g_b
6_on_b_i
6_on_h_c
6_stack_c_e
7_on_a_g
7_on_f_d
7_on_g_b
7_on_c_e
7_on_b_i
7_on_h_c
7_stack_i_k
8_on_a_g
8_on_f_d
8_on_g_b
8_on_c_e
8_on_b_i
8_on_h_c
8_on_i_k
8_stack_e_m
9_on_a_g
9_on_f_d
9_on_g_b
9_on_c_e
9_on_b_i
9_on_h_c
9_on_e_m
9_on_i_k
9_stack_k_f
10_on_a_g
10_on_f_d
10_on_k_f
10_on_g_b
10_on_c_e
10_on_b_i
10_on_h_c
10_on_e_m
10_on_i_k
10_stack_l_o
11_on_l_o
11_on_a_g
11_on_f_d
11_on_k_f
11_on_g_b
11_on_c_e
11_on_b_i
11_on_h_c
11_on_e_m
11_on_i_k
11_stack_j_h
12_on_l_o
12_on_a_g
12_on_f_d
12_on_k_f
12_on_g_b
12_on_c_e
12_on_b_i
12_on_h_c
12_on_e_m
12_on_i_k
12_on_j_h
12_stack_m_p
13_on_l_o
13_on_a_g
13_on_f_d
13_on_k_f
13_on_g_b
13_on_c_e
13_on_b_i
13_on_m_p
13_on_h_c
13_on_e_m
13_on_i_k
13_on_j_h
13_stack_q_n
14_on_l_o
14_on_a_g
14_on_f_d
14_on_k_f

14_on_g_b
14_on_c_e
14_on_b_i
14_on_m_p
14_on_q_n
14_on_h_c
14_on_e_m
14_on_i_k
14_on_j_h
14_stack_n_l
15_on_l_o
15_on_a_g
15_on_f_d
15_on_k_f
15_on_g_b
15_on_c_e
15_on_b_i
15_on_n_l
15_on_m_p
15_on_q_n
15_on_h_c
15_on_e_m
15_on_i_k
15_on_j_h
15_stack_p_a
16_on_l_o
16_on_a_g
16_on_f_d
16_on_k_f
16_on_g_b
16_on_c_e
16_on_p_a
16_on_b_i
16_on_n_l
16_on_m_p
16_on_q_n
16_on_h_c
16_on_e_m
16_on_i_k
16_on_j_h
16_stack_o_j
17_on_l_o
17_on_a_g
17_on_f_d
17_on_k_f
17_on_g_b
17_on_c_e
17_on_p_a
17_on_b_i
17_on_n_l
17_on_m_p
17_on_o_j
17_on_q_n
17_on_h_c
17_on_e_m
17_on_i_k
17_on_j_h

já a que demandou menor tempo e memoria foi a instancia “blocks-4-0.strips”, que demorou 1.01 segundos para ser executada, e utilizando 30.92 MB de memoria dividas e:

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    28     30.7 MiB     30.7 MiB           1   @profile
    29                                         def main():
    30     30.7 MiB      0.0 MiB           1       satPlanInstance = SatPlanInstance(sys.argv[1])
    31     30.7 MiB      0.0 MiB           1       instanceMapper  = SatPlanInstanceMapper()
    32     30.7 MiB      0.0 MiB           1       instanceMapper.add_list_of_literals_to_mapping(satPlanInstance.get_atoms())
    33     30.7 MiB      0.0 MiB           1       gluc = Glucose3()
    34     30.7 MiB      0.0 MiB           1       model= []
    35
    36     30.7 MiB      0.0 MiB           1       n_level = sys.maxsize   #MAIOR NUMERO INTEIRO, QUE REPRESENTA O N(INFINITO)
    37     30.8 MiB      0.0 MiB           4       for i in range(1, n_level): #UM LAÇOI DE RPETIÇÃO QUE VAI DO NIVEL 1 AO INFINITO
    38     30.8 MiB      0.0 MiB           4           a = create_literals_for_level_from_list(i, satPlanInstance.get_actions())
    39                                                 # print(a)
    40     30.8 MiB      0.0 MiB           4           instanceMapper.add_list_of_literals_to_mapping(a)
    41     30.8 MiB      0.0 MiB           4           actions_list = instanceMapper.get_list_of_literals_from_mapping(a)
    42                                                 # print(actions_list)
    43
    44     30.8 MiB      0.0 MiB         132           for actions in satPlanInstance.get_actions():
    45     30.8 MiB      0.0 MiB         128               pre_condition = create_literals_for_level_from_list(i, satPlanInstance.get_action_preconditions(actio
ns))
    46                                                     # CRIA OS LITERAIS DAS PRE CONDIÇÕES
    47     30.8 MiB      0.0 MiB         128               acoes_level= create_literal_for_level(i, actions)
    48                                                     #CRIA OS LITERAIS DAS AÇÕES POR LEVEL
    49     30.8 MiB      0.0 MiB         128               instanceMapper.add_list_of_literals_to_mapping(pre_condition)
    50                                                     #instanceMapper.add_list_of_literals_to_mapping(pre_condition)
    51     30.8 MiB      0.0 MiB         432               for pre in pre_condition:
    52     30.8 MiB      0.0 MiB         304                   mapped_condition = instanceMapper.get_literal_from_mapping(pre)   #MAPEIA AS PRE-CONDIÇÕES E MAND
```

```
    52     30.8 MiB      0.0 MiB         304                 mapped_condition = instanceMapper.get_literal_from_mapping(pre)   #MAPEIA AS PRE-CONDIÇÕES E MAND
A PARA O SOLVER
    53     30.8 MiB      0.0 MiB         304                 gluc.add_clause([-instanceMapper.get_literal_from_mapping(acoes_level), mapped_condition])
    54
    55     30.8 MiB      0.0 MiB         128             pos_condition = create_literals_for_level_from_list(i+1, satPlanInstance.get_action_posconditions(act
ions))
    56     30.8 MiB      0.0 MiB         128             instanceMapper.add_list_of_literals_to_mapping(pos_condition)
    57     30.8 MiB      0.0 MiB         736             for pos in pos_condition:   #MAPEIA AS POS-CONDIÇÕES E MANDA PRO SOLVER
    58     30.8 MiB      0.0 MiB         608                 mapped_condition2 = instanceMapper.get_literal_from_mapping(pos)
    59     30.8 MiB      0.0 MiB         608                 gluc.add_clause([-instanceMapper.get_literal_from_mapping(acoes_level), mapped_condition2])
    60
    61
    62                                                 # print(f'Inicial: {satPlanInstance.get_initial_state()}')
    63     30.8 MiB      0.0 MiB           4           initials_states = create_state_from_literals(satPlanInstance.get_initial_state(), satPlanInstance.get_sta
te_atoms())
    64     30.8 MiB      0.0 MiB           4           estado_inicial = create_literals_for_level_from_list(1, initials_states)
    65     30.8 MiB      0.0 MiB           4           instanceMapper.add_list_of_literals_to_mapping(estado_inicial)
    66     30.8 MiB      0.0 MiB         104           for estado in instanceMapper.get_list_of_literals_from_mapping(estado_inicial):
    67     30.8 MiB      0.0 MiB         100               gluc.add_clause([estado])
    68
    69                                                 # print(f'Final: {satPlanInstance.get_final_state()}')
    70     30.8 MiB      0.0 MiB           4           estado_final = create_literals_for_level_from_list(i, satPlanInstance.get_final_state())
    71                                                 # print(estado_final)
    72     30.8 MiB      0.0 MiB           4           instanceMapper.add_list_of_literals_to_mapping(estado_final)
    73
    74     30.8 MiB      0.0 MiB          16           for estado in instanceMapper.get_list_of_literals_from_mapping(estado_final):
    75     30.8 MiB      0.0 MiB          12               gluc.add_clause([estado])
    76
    77     30.9 MiB     -0.0 MiB          10           for j in range(1,i):
    78     30.9 MiB     -1.3 MiB         198               for acao in satPlanInstance.get_actions():
    79     30.9 MiB     -1.3 MiB         192                   poscondition = satPlanInstance.get_action_posconditions(acao)
```

```
    80     30.9 MiB     -1.3 MiB         192                   state= satPlanInstance.get_state_atoms()
    81     30.9 MiB    -33.8 MiB        4992                   for estado in state:
    82     30.9 MiB    -32.5 MiB        4800                       if estado not in poscondition:
    83                                                                 # AQUI MAPEIA OS LITERAIS POR NIVEL
    84     30.9 MiB    -29.3 MiB        4344                           level_estado = create_literal_for_level(j, estado)
    85     30.9 MiB    -29.3 MiB        4344                           level_acao = create_literal_for_level(j, acao)
    86     30.9 MiB    -29.3 MiB        4344                           level_prox_estado = create_literal_for_level(j + 1, estado)
    87
    88                                                                 # ADICIONA ELES AO SOLVER
    89     30.9 MiB    -29.2 MiB        4344                           gluc.add_clause([instanceMapper.get_literal_from_mapping(level_estado), -instanceMapper.get_li
teral_from_mapping(level_acao), -instanceMapper.get_literal_from_mapping(level_prox_estado)])
    90     30.9 MiB      0.0 MiB          10           for j in range(1, i):
    91     30.9 MiB      0.0 MiB           6               level_actions = create_literals_for_level_from_list(j, satPlanInstance.get_actions())
    92     30.9 MiB      0.0 MiB           6               mapped_actions = instanceMapper.get_list_of_literals_from_mapping(level_actions)
    93     30.9 MiB      0.0 MiB           6               gluc.add_clause(mapped_actions)
    94
    95
    96     30.9 MiB      0.0 MiB           4           gluc.solve()
    97     30.9 MiB      0.0 MiB           4           if gluc.get_model() != None:            #AQUI SE RESOLVIDO, PRINTA A RESOLUÇÃO
    98     30.9 MiB      0.0 MiB           1               print(f'NV{i}-resolvido')
    99     30.9 MiB      0.0 MiB           1               model = gluc.get_model()
   100     30.9 MiB      0.0 MiB         311               for j in list(gluc.get_model()):
   101     30.9 MiB      0.0 MiB         310                   if j > 0:
   102     30.9 MiB      0.0 MiB          18                       action = instanceMapper.get_literal_from_mapping_reverse(j)
   103     30.9 MiB      0.0 MiB          18                       print(instanceMapper.get_literal_from_mapping_reverse(j))
   104     30.9 MiB      0.0 MiB           1               break
   105                                                 else:
   106     30.9 MiB      0.0 MiB           3               print(f'NV{i}-não resolvido')
   107     30.8 MiB     -0.0 MiB           3               gluc.delete()
   108     30.8 MiB      0.0 MiB           3               gluc = Glucose3()
```

E seu resultado foi:
1_stack_c_b
1_clear_c
1_handempty
1_clear_b
1_clear_d
1_ontable_d
2_on_c_b
1_clear_a
1_ontable_a
1_ontable_c
1_ontable_b
2_stack_b_a
3_on_c_b
3_on_b_a
3_stack_d_c
4_on_c_b
4_on_b_a

4_on_d_c

Por fim, o tempo médio de todas as instancias executadas pelo SATSOLVE foi 15 minutos e 22 segundos, e a memoria media 57.944MB.

Conclusão

O desenvolvimento de uma solução computacional para o problema de ordenação de blocos no mundo dos blocos utilizando a biblioteca pysat em Python revelou-se desafiador, porém, ofereceu resultados promissores. A abordagem adotada, baseada na teoria dos algoritmos SAT e na implementação do SATSOLVER, permitiu encontrar soluções para diversas instâncias do problema, evidenciando a eficácia da proposta.

Os resultados dos experimentos destacaram a variabilidade no tempo de execução e no consumo de memória, de acordo com a complexidade da instancia passada com. A análise das instâncias extremas, desde a mais rápida até a mais demorada, mostrou que o contexto de cada instancia interfere diretamente no desempenho do algorítimo

A instância mais demorada, "blocks-17-0.strips", permite-nos extrair a capacidade de resolução de contextos mais complexos, apesar dos recursos utilizados serem maiores. Por outro lado, a instância mais rápida, "blocks-4-0.strips", demonstrou a eficiência do algoritmo em lidar com problemas menos complexos de forma ágil.

Por fim, o algorítimo implementado demonstra ser uma alternativa sólida para resolver o problema de ordenação de blocos no mundo dos blocos, proporcionando uma base para futuras otimizações e aprimoramentos.