Instituto Tecnológico de Aeronáutica

CTC-17: Inteligência Artificial

Projeto Buscas de Melhoria Iterativa e Satisfação de Restrição

Professor:
Paulo A L Castro

Grupo:
Eduardo A M Barbosa
Victor R Sales

17 de Setembro de 2019

Conteúdo

1 Objetivo do Trabalho e Descrição da Implementação

O objetivo do presente trabalho é explorar e fixar os conhecimentos obtidos sobre Melhorias Iterativa e sobre Problema da Satisfação de Restrições.

Para a implementação dos algoritmos de Melhoria Iterativa, foi utilizada a linguagem Python 3, mais especificadamente a versão 3.7.3, e não foi utilizada nenhuma IDE especializada, apenas o editor de texto Sublime.

2 Resultados Obtidos

2.1 Melhoria Iterativa

2.1.1 N-Rainhas

Para esse problema, decidimos implementar o algoritmo Random-restart Hill Climbling: rodamos o Simple Hill Climbing cinco vezes, cada uma com estados iniciais randômicos, e dizemos qual o melhor estado dentre os cinco atingidos.

Foi utilizada como função de otmização a quantidade distinta de linhas, colunas e diagonais principais e secundárias ocupadas por cada rainha. Ou seja, para N rainhas, o valor máximo dessa função seria $4N$, pois haveriam N linhas distintas sendo ocupadas, N colunas distintas sendo ocupadas e N de cada uma das diagonais.

Para fazer uma análise melhor do tempo, o algoritmo inteiro foi rodado 10 vezes para cada caso. Com isso, obtemos os resultados da Tabela 1.

Tabela 1: Tempos necessários para se obter a solução para cada número de rainhas considerado.

N	Tempo (s)
4	0.120 ± 0.010
5	0.256 ± 0.014
6	0.998 ± 0.887
7	2.087 ± 1.912
8	8.993 ± 11.976

Para os valores pedidos, não foi possível chegar numa solução em tempo hábil, apenas em uma aproximação. Para esses casos, o número máximo de iterações para o Simple Hill Climbing foi 100 e os resultados obtidos estão na Tabela 2.

Tabela 2: Tempos necessários para se obter uma solução aproximada para cada número de rainhas considerado.

N	Tempo (s)
10	1.036 ± 0.100
15	3.242 ± 0.155
20	7.378 ± 0.314
25	13.660 ± 0.897

2.1.2 Função Real

Para esse problema, foi implementado o algoritmo de Simulated Annealing, com a função de *schedule* sendo a função $s(t) = \frac{100}{t}$, com t começando de 1. Também, ao invés de infinitas iterações, utilizamos apenas 100.

Com isso, obtemos os máximos locais da Tabela 3.

Tabela 3: Máximos locais encontrados para a função real.

x	y	$f(x, y)$
0.00033	0.00110	3.99999

2.1.3 Discussão

Inicialmente utilizou-se uma mudança de estado para os lados e diagonais nos problemas das N-Rainhas. No entanto, observou-se que mudando a mudança de estado para apenas os lados o código executou de forma mais rápida e com precisão semelhante.

2.2 Satisfação de Restrições

2.2.1 Primeira Modelagem

Como sabemos, temos 5 casas adjacentes – enumere-mo-las de 1 a 5, com cada casa adjacente às casas com números adjacentes – cada uma possuindo 4 propriedades, sendo elas:

- C_i, a cor da casa i
- N_i, a nacionalidade de quem mora na casa i
- M_i, a marca do cigarro preferido por quem mora na casa i
- B_i, a bebida preferida de quem mora na casa i
- A_i, o animal de estimação de quem mora na casa i

O domínio de cada uma dessas variáveis é, respectivamente:

- {Vermelha, Amarela, Azul, Verde, Marfim}
- {Inglês, Espanhol, Norueguês, Ucraniano, Japonês}
- {Kool, Chesterfield, Winston, Lucky Strike, Parliament}
- {Água, Suco de Laranja, Chá, Café, Leite}
- {Zebra, Cachorro, Raposa, Caramujos, Cavalo}

De cara, como cada variável é única, já obtemos as seguintes retrições:

- $C_i \neq C_j, \forall i \neq j$
- $N_i \neq N_j, \forall i \neq j$
- $M_i \neq M_j, \forall i \neq j$
- $B_i \neq B_j, \forall i \neq j$
- $A_i \neq A_j, \forall i \neq j$

Ademais, os bullet points dados acarretam as restrições a seguir:

- $N_i = \text{Inglês} \iff C_i = \text{Vermelha}$
- $N_i = \text{Espanhol} \iff A_i = \text{Cachorro}$
- $N_1 = \text{Norueguês}$
- $C_i = \text{Amarela} \iff M_i = \text{Kool}$
- $M_i = \text{Chesterfield} \iff A_{i-1} = \text{Raposa} \vee A_{i+1} = \text{Raposa}$
- $N_i = \text{Norueguês} \iff C_{i-1} = \text{Azul} \vee C_{i+1} = \text{Azul}$
- $M_i = \text{Winston} \iff A_i = \text{Caramujos}$
- $M_i = \text{Lucky Strike} \iff B_i = \text{Suco de Laranja}$
- $N_i = \text{Ucraniano} \iff B_i = \text{Chá}$

- $N_i =$ Japonês $\iff M_i =$ Parliament
- $M_i =$ Kool $\iff A_{i-1} =$ Cavalo $\vee A_{i+1} =$ Cavalo
- $B_i =$ Café $\iff C_i =$ Verde
- $C_i =$ Verde $\iff C_{i-1} =$ Marfim
- $B_3 =$ Leite

2.2.2 Segunda Modelagem

Na modelagem anterior, indexamos nossas variáveis pela posição correspondente da casa. Para essa modelagem, vamos considerar as variáveis indexadas pelos domínios anteriormente dados e o domínio de cada variável será o domínio de indexação anterior (estamos revertendo o problema). Assim, temos as seguintes variáveis:

- C_c, a posição da casa de cor c
- N_n, a posição da casa onde mora o n
- M_m, a posição da casa onde se fuma m
- B_b, a posição da casa onde se bebe b
- A_a, a posição da casa onde se cria a

O domínio de cada uma dessas variáveis é o conjunto $D = \{1, 2, 3, 4, 5\}$ e os domínios de indexação são, respectivamente:

- {Vermelha, Amarela, Azul, Verde, Marfim}
- {Inglês, Espanhol, Norueguês, Ucraniano, Japonês}
- {Kool, Chesterfield, Winston, Lucky Strike, Parliament}
- {Água, Suco de Laranja, Chá, Café, Leite}
- {Zebra, Cachorro, Raposa, Caramujos, Cavalo}

Novamente, como cada variável é única, já obtemos as seguintes retrições:

- $C_i \neq C_j, \forall i \neq j$
- $N_i \neq N_j, \forall i \neq j$
- $M_i \neq M_j, \forall i \neq j$
- $B_i \neq B_j, \forall i \neq j$

- $A_i \neq A_j, \forall i \neq j$

Ademais, os bullet points dados acarretam as restrições a seguir:

- $N_{\text{Inglês}} = C_{\text{Vermelha}}$
- $N_{\text{Espanhol}} = A_{\text{Cachorro}}$
- $N_{\text{Norueguês}} = 1$
- $C_{\text{Amarela}} = M_{\text{Kool}}$
- $|M_{\text{Chesterfield}} - A_{\text{Raposa}}| = 1$
- $|N_{\text{Norueguês}} - C_{\text{Azul}}| = 1$
- $M_{\text{Winston}} = A_{\text{Caramujos}}$
- $M_{\text{Lucky Strike}} = B_{\text{Suco de Laranja}}$
- $N_{\text{Ucraniano}} = B_{\text{Chá}}$
- $N_{\text{Japonês}} = M_{\text{Parliament}}$
- $|M_{\text{Kool}} - A_{\text{Cavalo}}| = 1$
- $B_{\text{Café}} = C_{\text{Verde}}$
- $C_{\text{Verde}} = C_{\text{Marfim}} + 1$
- $B_{\text{Leite}} = 3$

2.2.3 Discussão

A primeira modelagem foi a que pareceu mais óbvia no começo e foi fácil de traduzir a linguagem natural para a linguagem lógica apropriada para o modelamento. Mas o grande problema é que as restrições acabaram por ficar complicadas de mais, com 3 delas envolvendo 3 variáveis, o que torna o problema um possível 3-Sat e, portanto, NP-Completo para se resolver. Além disso, teria que se tomar cuidado com corner cases na implementação dessas restrições que checam vizinhos.

A segunda modelagem é mais robusta, pois não necessita de tratamento de corner cases e todas as restrições são de uma ou duas variáveis apenas, o que torna torna a solução possivelmente muito mais rápida. Além disso, todas elas são checks simples e pragmáticos de serem feitos por uma linguagem de programação, o que torna esta modelagem a mais simples e efetiva de ser programada e, consequentemente, a preferível.

3 Conclusões

O trabalho mostrou-se desafiador e encontramos dificuldades em todos os itens pedidos. Para o problema das N-Rainhas, em particular, demorava-se um tempo muito grande para se achar uma solução, tanto que nosso algoritmo não chegava à melhor solução para nenhum dos casos pedidos (não nos 10 minutos que deixamos rodando para as 10 rainhas) e usamos apenas uma solução aproximada nas respostas.