Projeto 2 – Flying Tourist Problem

Algoritmos para Lógica Computacional

2024/2025

Afonso da Conceição Ribeiro (ist1102763) – Grupo **20**

Instituto Superior Técnico – Universidade de Lisboa

Índice

1. Problema a resolver

O Flying Tourist Problem é um problema de otimização em que um turista precisa de planear uma viagem por várias cidades, minimizando o custo dos bilhetes de avião. Este inicia e termina a viagem na mesma cidade, e o número de noites que passa em cada cidade pertence a um intervalo definido para cada uma. A ordem das cidades a visitar não é predefinida, e o turista é obrigado a utilizar apenas voos diretos. O problema é modelado como um problema de Satisfiability Modulo Theories (SMT) e resolvido utilizando um solver correspondente.

2. Como instalar e correr o projeto

- Clonar o repositório do GitLab.
- Navegar até à diretoria do projeto.
- Correr o projeto utilizando o seguinte comando:
  `python3 project2.py < input.ttp > output.myout`
- Comparar o conteúdo do ficheiro de output obtido com o output esperado.

3. Codificações experimentadas para o problema

A partir de um ficheiro de input que represente uma instância do problema, obtêm-se as seguintes variáveis e conjuntos:

- $\mathcal{C}$: conjunto das cidades que o turista pretende visitar e a cidade de origem; $n = |\mathcal{C}|$
- $base$: a cidade de origem, $base \in \mathcal{C}$
- k_c^m: número mínimo de noites a passar na cidade c, $\forall c \in \mathcal{C} \setminus \{base\}$
- k_c^M: número máximo de noites a passar na cidade c, $\forall c \in \mathcal{C} \setminus \{base\}$
- $\mathcal{F}$: conjunto dos voos em consideração; $m = |\mathcal{F}|$
- d_f: data do voo f, $\forall f \in \mathcal{F}$
- w_f: custo do voo f, $\forall f \in \mathcal{F}$

E definem-se as seguintes:

- $x_i = 1$ se e só se o voo f_i é escolhido
- k_c: número efetivo de noites a passar na cidade c, $\forall c \in \mathcal{C} \setminus \{base\}$
- $K = d_{f_m} - d_{f_1}$: número de noites entre a primeira e a última datas com voos
- $K_m = \sum_{c \in \mathcal{C}} k_c^m$: número mínimo de noites a viajar
- $\mathcal{O}_c \subset \mathcal{F}$: conjunto dos voos com origem na cidade c, $\forall c \in \mathcal{C}$
- $\mathcal{D}_c \subset \mathcal{F}$: conjunto dos voos com destino à cidade c, $\forall c \in \mathcal{C}$

Os conjutos de cláusulas utilizadas nas codificações do problema são descritos nas subsecções seguintes.

3.1. Com filtragem de voos compatíveis

Pretende-se minimizar o valor da seguinte expressão:

$$\sum_{i=1}^{m} x_i \cdot w_i \tag{1}$$

O primeiro conjunto de cláusulas garante que, para cada cidade, são escolhidos exatamente um voo com destino e um com origem na mesma.

$$\bigwedge_{c \in \mathcal{C}} \left(\sum_{\substack{i=1 \\ f_i \in \mathcal{D}_c}}^{m} x_i = 1 \wedge \sum_{\substack{i=1 \\ f_i \in \mathcal{O}_c}}^{m} x_i = 1 \right) \tag{2}$$

O segundo conjunto de cláusulas assegura que, para cada cidade visitada, se um certo voo com destino nessa cidade é escolhido, então o voo com origem nessa cidade escolhido acontece entre k_c^m e k_c^M dias depois.

$$\bigwedge_{c \in \mathcal{C} \setminus \{base\}} \left(\bigwedge_{\substack{i=1 \\ f_i \in \mathcal{D}_c}}^{m} \left(x_i \Rightarrow \bigvee_{\substack{j=1 \\ f_j \in \mathcal{O}_c \\ k_c^m \leq d_{f_j} - d_{f_i} \leq k_c^M}}^{m} x_j \right) \right) \tag{3}$$

O terceiro conjunto de cláusulas é semelhante ao anterior, garantindo que, para cada cidade visitada, se um certo voo com origem nessa cidade é escolhido, então o voo com destino nessa cidade escolhido acontece entre k_c^m e k_c^M dias antes. Este conjunto de cláusulas é redundante, pois a sua semântica é já garantida por φ_2, mas a sua presença é relevante porque reduz o espaço de procura, aumentando a eficiência do programa.

$$\bigwedge_{c \in \mathcal{C} \setminus \{base\}} \left(\bigwedge_{\substack{i=1 \\ f_i \in \mathcal{O}_c}}^{m} \left(x_i \Rightarrow \bigvee_{\substack{j=1 \\ f_j \in \mathcal{D}_c \\ k_c^m \leq d_{f_i} - d_{f_j} \leq k_c^M}}^{m} x_j \right) \right) \tag{4}$$

No caso da cidade *base*, é o voo com destino na cidade que tem de acontecer depois do voo com origem, mais especificamente entre K_m e K dias depois. Isto é cumprido pelo seguinte conjunto de cláusulas, análogo aos últimos dois para a situação específica da cidade *base*.

$$\bigwedge_{\substack{i=1 \\ f_i \in \mathcal{O}_{base}}}^{m} \left(x_i \Rightarrow \bigvee_{\substack{j=1 \\ f_j \in \mathcal{D}_{base} \\ K_m \leq d_{f_j} - d_{f_i} \leq K}}^{m} x_j \right) \wedge \bigwedge_{\substack{i=1 \\ f_i \in \mathcal{D}_{base}}}^{m} \left(x_i \Rightarrow \bigvee_{\substack{j=1 \\ f_j \in \mathcal{O}_{base} \\ K_m \leq d_{f_i} - d_{f_j} \leq K}}^{m} x_j \right) \tag{5}$$

Finalmente, a seguinte cláusula não afeta o espaço de procura, estando presente apenas com o objetivo de forçar o solver a resolver o problema utilizando a teoria de Aritmética Linear de Inteiros.

$$\sum_{c \in \mathcal{C}} k_c \leq K \tag{6}$$

3.2. Com variáveis para os dias de chegada e de partida de cada cidade

Obtêm-se, adicionalmente, as seguintes variáveis:

- c_f^O: cidade de origem do voo f, $\forall f \in \mathcal{F}$
- c_f^D: cidade de destino do voo f, $\forall f \in \mathcal{F}$

Definem-se as seguintes novas variáveis inteiras:

- d_c^C: data de chegada à cidade c, $\forall c \in \mathcal{C} \setminus \{base\}$
- d_c^P: data de partida da cidade c, $\forall c \in \mathcal{C} \setminus \{base\}$

Pretende-se minimizar o valor da seguinte expressão:

$$\sum_{i=1}^{m} x_i \cdot w_i \tag{7}$$

O primeiro conjunto de cláusulas estabelece os limites das datas de chegada e partida de cada cidade, que devem estar entre o primeiro e o último dias em que existem voos.

$$\bigwedge_{c \in \mathcal{C}} \left(\left(d_{f_1} \leq d_c^C \leq d_{f_m} \right) \wedge \left(d_{f_1} \leq d_c^P \leq d_{f_m} \right) \right) \tag{8}$$

O segundo conjunto de cláusulas garante que, para cada cidade, são escolhidos exatamente um voo com destino e um com origem na mesma.

$$\bigwedge_{c \in \mathcal{C}} \left(\sum_{\substack{i=1 \\ f_i \in \mathcal{D}_c}}^{m} x_i = 1 \wedge \sum_{\substack{i=1 \\ f_i \in \mathcal{O}_c}}^{m} x_i = 1 \right) \tag{9}$$

O terceiro conjunto de cláusulas assegura que, se um voo é escolhido, a data do voo é igual à data de partida da sua cidade de origem e à data de chegada à sua cidade de destino.

$$\bigwedge_{f_i \in \mathcal{F}} \left(x_i \Rightarrow \left(d^C_{c^D_{f_i}} = d_{f_i} \wedge d^P_{c^O_{f_i}} = d_{f_i} \right) \right) \tag{10}$$

O quarto conjunto de cláusulas garante que o número de noites passado em cada cidade está dentro dos limites.

$$\bigwedge_{c \in \mathcal{C}} C \left(\left(k_c^m \leq d_c^P - d_c^C \right) \wedge \left(d_c^P - d_c^C \leq k_c^M \right) \right) \tag{11}$$

O quinto conjunto de cláusulas garante que, para cada par de cidades visitadas diferentes, estas não têm estadias sobrepostas.

$$\bigwedge_{\substack{c_1 \in \mathcal{C} \setminus \{base\} \\ c_2 \in \mathcal{C} \setminus \{base\} \\ c_1 \neq c_2}} \left(\left(d_{c_1}^P <= d_{c_2}^C \right) \vee \left(d_{c_2}^P <= d_{c_1}^C \right) \right) \tag{12}$$

Finalmente, o sexto conjunto de cláusulas assegura que, nos dias de partida e de chegada à cidade *base*, há, respetivamente, chegada e partida a exatamente uma outra cidade. Note-se que a comparação de duas variáveis é um valor booleano, correspondente aos inteiros 1 ou 0 caso seja, respetivamente, verdadeiro ou falso.

$$\left(\sum_{c \in \mathcal{C}} \left(d_{base}^P = d_c^C \right) = 1 \right) \wedge \left(\sum_{c \in \mathcal{C}} \left(d_{base}^C = d_c^P \right) \right) = 1 \tag{13}$$

4. Algoritmo e configurações usadas

O solver é uma instância da classe `Optimize`, da biblioteca `z3`. Vão-lhe sendo adicionadas cláusulas utilizando o método `append` e, no fim, este contém a fórmula SMT que codifica o problema, chamando-se o método `check` para verificar se a fórmula é satisfazível e, caso seja, o método `model` devolve uma atribuição das variáveis da fórmula.

Finalmente, encontram-se as variáveis x_i às quais foi atribuído o valor `True`, que correspondem aos voos escolhidos, e calcula-se a soma dos custos desses voos. Finalmente, imprimem-se o valor total e os voos escolhidos para o `stdout`, no formato requerido.

5. Avaliação experimental e análise das codificações experimentadas

As seguintes tabelas apresentam os tempos, em segundos, que cada uma das codificações leva a executar os testes públicos. A indicação `TLE` (Time Limit Exceeded) significa que o programa não conseguiu chegar à solução dentro do limite de tempo definido (10 minutos).

	T01	T02	T03	T04	T05	T06	T07	T08	T09
3.1.	0.09	0.06	0.05	0.03	0.05	0.09	0.11	0.14	0.21
3.2.	0.10	0.07	0.07	0.12	0.09	0.27	0.48	1.04	2.49

Table 1: Tempos de execução dos testes públicos 1 a 9

	T10	T11	T12	T13	T14	T15	T16	T17	T18
3.1.	0.41	0.40	3.95	0.56	1.47	4.27	2.61	0.69	0.79
3.2.	28.14	14.96	193.59	101.00	216.54	598.75	`TLE`	240.99	575.30

Table 2: Tempos de execução dos testes públicos 10 a 18

Analisando estes dados, verifica-se que a codificação 3.1. levou menos tempo do que a codificação 3.2. a executar todos os testes, sendo bastante mais eficiente. Tal facto revela que a codificação 3.1. é melhor a limitar o espaço de procura, chegando mais rápido a uma solução.