Projeto 1 – 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 passa em cada cidade um número fixo de noites. 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 otimização de Maximum Satisfiability (MaxSAT) 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 project1.py < input.ttp > output.myout`
- Comparar o conteúdo do ficheiro de output obtido com o output esperado.

3. Codificação do 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
- $n = |\mathcal{C}|$: número total de cidades
- $base$: a cidade de origem, $base \in \mathcal{C}$
- k_c: número de noites que o turista pretende passar na cidade c, $\forall c \in \mathcal{C} \setminus \{base\}$
- $\mathcal{F}$: conjunto dos voos em consideração
- $m = |\mathcal{F}|$: número total de voos
- 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
- $K = \sum_{i=1, c_i \neq base}^{n} k_{c_i}$
- $\mathcal{O}_c \subset \mathcal{F}$: 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 na codificação do problema são descritos nas subsecções seguintes.

3.1. Soft clauses

O conjunto das soft clauses, φ_S, contém as negações das escolhas de cada um dos voos, com peso associado w_{f_i}, para minimizar o custo da viagem.

$$\varphi_S = \bigwedge_{i=1}^{m} \neg x_i \tag{1}$$

3.2. Hard clauses

O conjunto das hard clauses, φ_H, é definido da seguinte forma, sendo os conjuntos φ_1, φ_2, φ_3 e φ_4 definidos a seguir.

$$\varphi_H = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \tag{2}$$

Os primeiros dois conjuntos de hard clauses, φ_1 e φ_2, garantem que, para cada cidade, há exatamente um voo escolhido com, respetivamente, destino e origem na mesma.

$$\varphi_1 = \bigwedge_{c \in \mathcal{C}} \left(\sum_{\substack{i=1 \\ f_i \in \mathcal{D}_c}}^{m} x_i = 1 \right) \tag{3}$$

$$\varphi_2 = \bigwedge_{c \in \mathcal{C}} \left(\sum_{\substack{i=1 \\ f_i \in \mathcal{O}_c}}^{m} x_i = 1 \right) \tag{4}$$

De seguida, o conjunto de hard clauses φ_3 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 k_c dias depois (note-se que $x \Rightarrow y \equiv \neg x \vee y$).

$$\varphi_3 = \bigwedge_{c \in \mathcal{C} \setminus \{base\}} \left(\bigwedge_{\substack{i=1 \\ f_i \in \mathcal{D}_c}}^{m} \left(\neg x_i \vee \bigvee_{\substack{j=1 \\ f_j \in \mathcal{O}_c \\ d_{f_j} = d_{f_i} + k_c}}^{m} x_j \right) \right) \tag{5}$$

No caso da cidade *base*, é o voo com destino na cidade que tem de acontecer depois do voo com origem, mais especificamente K dias depois, o que é cumprido pelo conjunto de hard clauses φ_4.

$$\varphi_4 = \bigwedge_{\substack{i=1 \\ f_i \in \mathcal{O}_{base}}}^{m} \left(\neg x_i \vee \bigvee_{\substack{j=1 \\ f_j \in \mathcal{D}_{base} \\ d_{f_j} = d_{f_i} + K}}^{m} x_j \right) \tag{6}$$

Desta forma, fica definida a fórmula MaxSAT que codifica o problema: $\varphi = \varphi_S \wedge \varphi_H$.

4. Algoritmo e configurações usadas

A fórmula MaxSAT que codifica o problema é uma instância da classe `WCNF`, da biblioteca `PySAT`, que permite manipular fórmulas CNF com soft clauses e hard clauses, e é construída utilizando os métodos `append` ou `extend` para adicionar uma cláusula ou um conjunto de cláusulas, respetivamente.

O algoritmo utilizado é o `RC2`, da biblioteca `PySAT`, cuja instância é criada recebendo no seu construtor a fórmula `WCNF`. O seu método `compute` calcula a solução ótima e devolve uma lista das variáveis que constam na fórmula, positivas caso lhes tenha sido atribuído o valor `True` e negativas caso contrário. Para tal, o `RC2` pode usar qualquer SAT solver disponível no `PySAT`, usando, neste caso, por defeito, o `g3`.

De seguida, encontram-se as variáveis positivas da solução, 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.

4.1. Formulação das cláusulas

A formulação das cláusulas correspondentes às fórmulas lógicas 1, 5 e 6 é direta, pois estas apenas utilizam os conectores lógicos $\wedge$ e $\vee$, sendo que, em Python, as disjunções são representadas através de listas de variáveis, enquanto as conjunções são tratadas ao adicionar múltiplas cláusulas à fórmula.

Já a formulação das cláusulas correspondentes às fórmulas 3 e 4 utiliza restrições Equals1, pelo que se utiliza o método `equals` da classe `CardEnc` da biblioteca `PySAT`, o qual devolve um conjunto de cláusulas CNF construídas a partir de um tipo de codificação especificado, neste caso, pairwise.

4.1.1. Tipo de codificação pairwise

A codificação de restrições Equals1 é feita a partir da conjunção das restrições AtMost1 e AtLeast1:

$$\sum_{i=1}^{n} x_i = 1 \quad \equiv \quad \left(\sum_{i=1}^{n} x_i \leq 1\right) \wedge \left(\sum_{i=1}^{n} x_i \geq 1\right) \tag{7}$$

A restrição AtLeast1 assegura que pelo menos uma das variáveis tem atribuído o valor `True`, o que se obtém a partir da disjunção de todas:

$$\sum_{i=1}^{n} x_i \geq 1 \quad \equiv \quad (x_1 \vee \cdots \vee x_n) \tag{8}$$

A restrição AtMost1 garante que, no máximo, uma das variáveis é atribuída a `True`. Utilizando a codificação pairwise, criam-se ${}^{n}C_2$ cláusulas, uma para cada par de variáveis distintas, sendo que o significado de cada cláusula é que, para um par de variáveis, pelo menos uma delas deve ser falsa. Se fossem atribuídos valores `True` a duas variáveis diferentes, haveria uma cláusula contendo ambas como verdadeiras, o que violaria a restrição. Portanto, a codificação pairwise garante que apenas uma variável será verdadeira.

$$\sum_{i=1}^{n} x_i \leq 1 \quad \equiv \quad \bigwedge_{1 \leq i < j \leq n} (\neg x_i \vee \neg x_j) \tag{9}$$

5. Variantes deste problema

5.1. Com escalas diurnas ao viajar entre cidades

A partir do ficheiro de input, obtêm-se agora, também, as horas de partida e de chegada dos voos, não necessárias anteriormente:

- h_f^d: hora de partida do voo f, $\forall f \in \mathcal{F}$
- h_f^a: hora de chegada do voo f, $\forall f \in \mathcal{F}$

Definem-se as seguintes variáveis e conjuntos:

- $\omega \subset \mathcal{F}$: um itinerário é uma lista ordenada de voos, tal que:
  - $|\omega| = \lambda$; se $\lambda = 1$, ω é apenas um voo
  - ω_i é o i-ésimo voo de ω
  - $\forall i \in \{1, \ldots, \lambda - 1\}: \quad \omega_i \in \mathcal{D}_c \Rightarrow \omega_{i+1} \in \mathcal{O}_c \quad , \quad d_{\omega_i} = d_{\omega_{i+1}} \quad , \quad h^a_{\omega_i} \leq h^d_{\omega_{i+1}}$
  - $\omega \in \mathcal{O}_c \Leftrightarrow \omega_1 \in \mathcal{O}_c$
  - $\omega \in \mathcal{D}_c \Leftrightarrow \omega_\lambda \in \mathcal{D}_c$
- $\mathcal{I} \subset \mathcal{P}(\mathcal{F})$: conjunto dos itinerários em consideração

E alteram-se as definições das seguintes variáveis e conjuntos já usados:

- d_ω: data do itinerário ω
- $w_\omega = \sum_{i=1}^{\lambda} w_{\omega_i}$: custo do itinerário ω
- $x_i = 1$ se e só se o itinerário ω_i é escolhido
- $\mathcal{O}_c \subset \mathcal{I}$: conjunto dos itinerários com origem na cidade c, $\forall c \in \mathcal{C}$
- $\mathcal{D}_c \subset \mathcal{I}$: conjunto dos itinerários com destino à cidade c, $\forall c \in \mathcal{C}$

Dadas estas mudanças, os conjuntos de cláusulas utilizadas na codificação do problema mantêm-se, passando-se a usar o conceito de itinerário no lugar do conceito de voo. Um itinerário consiste numa viagem entre duas cidades utilizando um ou mais voos, todos no mesmo dia (ou seja, com possíveis escalas diurnas). Generalizando o conceito de voo para itinerário com cardinalidade 1, obtém-se uma solução correspondente a um conjunto de itinerários, que por sua vez permite chegar ao conjunto de voos escolhidos.

5.2. Com um número de noites flexível entre cidades

A partir do ficheiro de input, obtêm-se agora, em vez de k_c, os números mínimos e máximos de noites em cada cidade:

- k_c^-: número mínimo de noites a passar na cidade c, $\forall c \in \mathcal{C} \setminus \{base\}$
- k_c^+: número máximo de noites a passar na cidade c, $\forall c \in \mathcal{C} \setminus \{base\}$

E definem-se as seguintes variáveis:

- $K^- = \sum_{i=1}^{n} k_{c_i}^-$
- $K^+ = \sum_{i=1}^{n} k_{c_i}^+$

É necessário proceder a alterações nas fórmulas lógicas 5 e 6:

$$\varphi_3 = \bigwedge_{c \in \mathcal{C} \setminus \{base\}} \left(\bigwedge_{\substack{i=1 \\ f_i \in \mathcal{D}_c}}^{m} \left(\neg x_i \vee \bigvee_{\substack{j=1 \\ f_j \in \mathcal{O}_c \\ k_c^- \leq d_{f_j} - d_{f_i} \leq k_c^+}}^{m} x_j \right) \right) \tag{10}$$

$$\varphi_4 = \bigwedge_{\substack{i=1 \\ f_i \in \mathcal{O}_{base}}}^{m} \left(\neg x_i \vee \bigvee_{\substack{j=1 \\ f_j \in \mathcal{D}_{base} \\ K^- \leq d_{f_j} - d_{f_i} \leq K^+}}^{m} x_j \right) \tag{11}$$

Note-se que estas restrições na escolha de voos garantem que se continua a ter, para os números efetivos de noites a passar fora de *base* e em cada cidade, $K = \sum_{i=1, c_i \neq base}^{n} k_{c_i}$.