INF05010 Otimização Combinatória

Algoritmo de Simulated Annealing para o Problema das Viagens de Avião

Ernesto Vaz de Oliveira e Ricco Vasconcellos Constante Soares

1. Introdução

O objetivo do presente trabalho foi implementar a meta-heurística de Simulated Annealing para resolver o problema das Viagens de Avião.

O problema das Viagens de Avião pode ser descrito da seguinte maneira: dados **k** aviões e **n** pessoas, conhecendo a capacidade **P** máxima de cada avião, o preço **ci**, **i** ∈ **[n]** que cada pessoa está disposta a pagar pela passagem, além de um preço adicional **Cij, i** ∈ **[n], j** ∈ **[n]** para cada outra pessoa caso elas sejam alocadas no mesmo avião, deseja-se obter a configuração das pessoas alocadas em aviões que maximiza o lucro

2. Formulação do Programa Inteiro

Uma solução de VA será representada por uma matriz *Vij*, com *i*∈*[k]* e *j*∈*[n]*, possuindo valor um se pessoa *j* estará em avião *i*, caso contrário valor será zero.

Função Objetivo:

Maximiza: $$\sum_{i=1}^{k}(\sum_{j=1}^{n}(c_j \times V_{ij} + \sum_{o=1}^{n} c_{oj} \times V_{io} \times V_{ij}))$$

Sujeito a:

$$\sum_{i=1}^{k} V_{ij} \leq 1, \forall j \in [n](1)$$

$$\sum_{j=1}^{n} p_j \times V_{ij} \leq P_i, \forall i \in [k](2)$$

$$V_{ij} \in \{0,1\}, \forall j \in [n], \forall i \in [k](3)$$

(1) Nenhuma pessoa irá viajar em mais de 1 avião.
(2) Somatório dos pesos das pessoas não excede capacidade do avião.
(3) A matriz de solução só assume valores booleanos.

3. Simulated annealing como meta-heurística

A solução abordada foi a de utilizar como meta-heurística o algoritmo de simulated annealing, que, através de uma analogia com o recozimento na metalurgia, fundamenta um processo de busca local no espaço do problema que diminui sua probabilidade de aceitar soluções piores ao longo do tempo. Com isso, buscou-se estabelecer uma vizinhança para soluções do problema e realizar uma busca local por máximos usando o algoritmo de simulated annealing.

4. Parâmetros

Além de uma instância do problema, o algoritmo de simulated annealing como meta-heurística recebe uma série de parâmetros: taxa de decaimento *r*, número de iterações em temperatura constante *I*, temperatura inicial *Ti* e um valor *k* de iterações para o algoritmo de Metropolis. Outro possível parâmetro seria a função de probabilidade de aceitação. Contudo, será adotada a escolha convencional de usar a distribuição de Boltzmann para essa probabilidade.

Para a **temperatura inicial**, foram geradas soluções aleatórias (inserindo o maior número de pessoas possível em cada avião de forma aleatória) e a temperatura inicial é igual ao valor da função objetivo da pior solução aleatória, subtraído da melhor. Isso foi feito de forma a gerar temperatura com grandeza próxima à magnitude da função objetivo.

O **número de iterações sem alterar temperatura** (parâmetro I) deve ter magnitude próxima à da vizinhança do problema, portanto foi tomado I como *n*k*. Esse valor é proporcional ao tamanho da vizinhança, pois cada solução pode possuir até *nk* vizinhos, alterando os *nk* valores de V.

Para os seguintes parâmetros foram feitos testes para encontrar valores ideais experimentalmente:

- **Taxa de resfriamento** dentro do segmento [0.8, 0.99], seguindo a implementação original de Kirkpatrick et al (1983).
- **Número de iterações** do algoritmo de Metropolis dentro do segmento [1, 20], de forma a não elevar excessivamente o tempo de execução.
- **Temperatura mínima** dentro do segmento [0.1, 1000].

Solução inicial

A solução inicial foi gerada usando um algoritmo guloso, pois permitirá encontrar soluções razoáveis com um método simples.

Proposta de algoritmo guloso:

1. Inicializa vetor de espaço disponível de aviões *espaçoDisponível[]* com os *k* pesos máximos de *P*. Inicializa matriz de solução *V* com valor 0 em todas as posições *Vij*.
2. **Loop**:
    - **2.1.** **Para** cada avião:
        - 2.2. Seleciona passageiro *i* com maior proporção *ci/pi*, tal que *pi* <= *espaçoDisponível[k]*.
            - 2.2.1. Se não houver, **termina**.
        - 2.3. Insere *i* em *k* (*Vki = 1)* e atualiza *espaçoDisponível[k]* subtraindo *pi* desse valor.
        - 2.4. Enquanto houver passageiro *j* tal que *pj* <= *espaçoDisponível[k]*

2.4.1. Seleciona passageiro *j* com maior razão *(cj + cji)/pj.*

2.4.2. Insere *j* em *k* (*Vkj = 1*) e atualiza *espaçoDisponível[k]* subtraindo *pj* desse valor.

5. Algoritmo

Algoritmo 1 - Algoritmo de Metropolis, recebe *solução_atual, temperatura* e *k*.

melhor_solução = solução_atual
Para *iteração* **de** 0 **até** *k*:
    candidata = solução_atual
    candidata = vizinho aleatório de *candidata*
    delta = candidata.valor - solução_atual.valor
    Se *delta* > 0:
        solução_atual = candidata
        Se *solução_atual.valor > melhor_solução.valor*:
            melhor_solução = solução_atual
    Senão se *delta* < 0:
        probabilidade_de_aceitação = $e^{\frac{-delta}{temperatura}}$
        com probabilidade *probabilidade_de_aceitação* **faz**:
            solução_atual = candidata
Retorna *melhor_solução*

Algoritmo 2 - Algoritmo de Simulated Annealing

Enquanto (*temperatura > temperatura_mínima*):
    Para *iteração* **de** 0 **até** *I*:
        candidata = **metropolis**(*solução_atual, temperatura, k*)
        delta = *candidata.valor - solução_atual.valor*
        Se *delta* > 0:
            solução_atual = candidata
    *temperatura = temperatura * resfriamento*
Retorna *solução_atual*

5.1. Metropolis

Para o algoritmo usado na meta-heurística para busca local, foi utilizado o algoritmo de Metropolis, de forma a amostrar aleatoriamente o espaço de soluções de problema, mantendo armazenada a melhor solução, de forma a não retornar soluções piores do que a atual.

Para isso, foi implementado método que transiciona solução para um vizinho aleatório, escolhendo um valor aleatoriamente da matriz de solução e invertendo-o, caso crie solução factível. Caso não, o método seleciona outro valor aleatório até encontrar um vizinho factível.

5.2. Critério de terminação

Seguindo a ideia do algoritmo de simulated annealing, a implementação utilizou como condição de parada uma **temperatura mínima.** A motivação é que com o resfriamento do recozimento, em temperaturas baixas o algoritmo aceita apenas soluções melhores, tornando-se um algoritmo de *hill climbing*, e portanto tenderá a estabilizar.

6. Implementação

6.1. Plataforma de Implementação

O trabalho foi testado em um sistema operacional *Arch Linux x86_64* com um processador Intel(R) Core(TM) i3-6100 com dois núcleos físicos e 4 virtuais de 3.7GHz, com 8GB de memória. A linguagem de programação utilizada foi Python3. Para a solução dos valores ótimos foi utilizado o solver GLPK na linguagem de programação Julia através do kernel IJulia.

6.2. Estruturas de dados

A matriz V foi representada na implementação através de um array bidimensional, de tamanho (k,n).

De forma a otimizar operações de testar factibilidade, a estrutura de dados relacionada a soluções possui também um array de tamanho k que representa o peso "disponível" em cada avião. De mesmo modo, a estrutura também contém um valor inteiro representando o valor objetivo da solução, assim, soluções vizinhas não precisam ser reavaliadas, visto que a vizinhança permite que o valor objetivo seja calculado somando ou subtraindo o valor alterado.

6.3. Avaliação de soluções

Dada a presença do campo de valor na estrutura de dados da solução, a avaliação de soluções consistiu em atualizar o valor da estrutura. Para isso, soluções são inicializadas com valor zero e foi implementado um método de alocação de pessoa a avião, que soma ao valor da solução o valor individual atrelado à pessoa e também os valores extras relacionados às pessoas de mesmo avião.

7. Teste de Parâmetros

Três parâmetros do Simulated Annealing foram variados; a temperatura mínima, taxa de resfriamento e parâmetro k (do algoritmo metrópolis). Os valores padrões assumidos para esses parâmetros (quando não foram variados) foram, respectivamente, 10, 0.9, 10. Cada teste foi realizado 5 vezes, utilizando uma semente randômica diferente em cada uma, de modo que os valores mostrados correspondem à média dos valores obtidos nas 5 execuções.

7.1. Parâmetro Temperatura Mínima

O parâmetro de temperatura mínima do Simulated Annealing foi variado, assumindo os valores [0.1,1,10,50,100,200,300,400,500,600,700,800,900,1000]. A Figura 1 mostra os resultados.

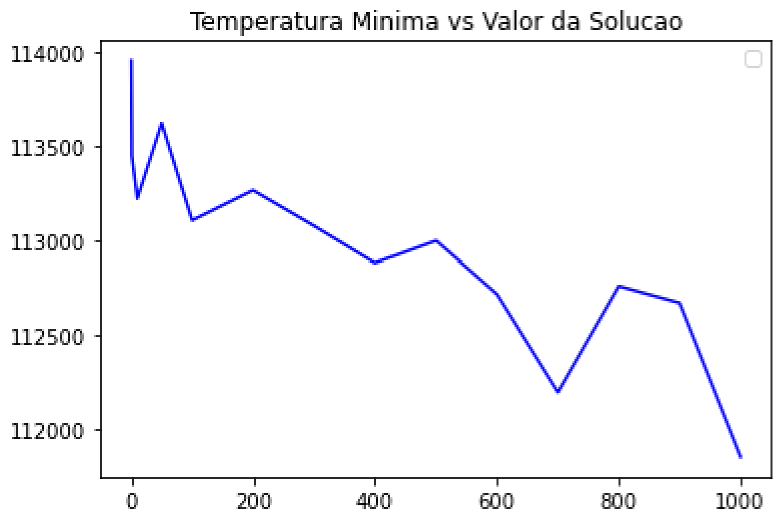

Figura 1: Variação da temperatura mínima

Como podemos observar, a alteração do parâmetro impacta significativamente na qualidade das soluções obtidas.

7.2. Parâmetro Taxa de Resfriamento

A taxa de resfriamento da temperatura do algoritmo foi variada, assumindo valores entre 0.8 e 0.9, passo 0.1. Os resultados são exibidos na Figura 2.

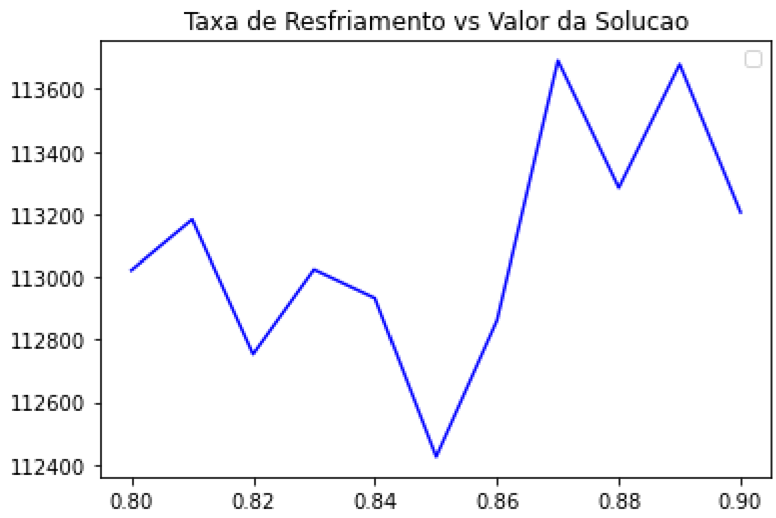

Figura 2: Variação da taxa de resfriamento

Os resultados indicam que esse é o parâmetro que menos influencia o valor da solução encontrada. Uma possível explicação para tal é que o intervalo testado constitui uma faixa de valores condizentes para o parâmetro.

7.3. Parâmetro K

O parâmetro k do algoritmo Metropolis foi alterado de 1 a 20, passo 1. Os resultados são mostrados na Figura 3.

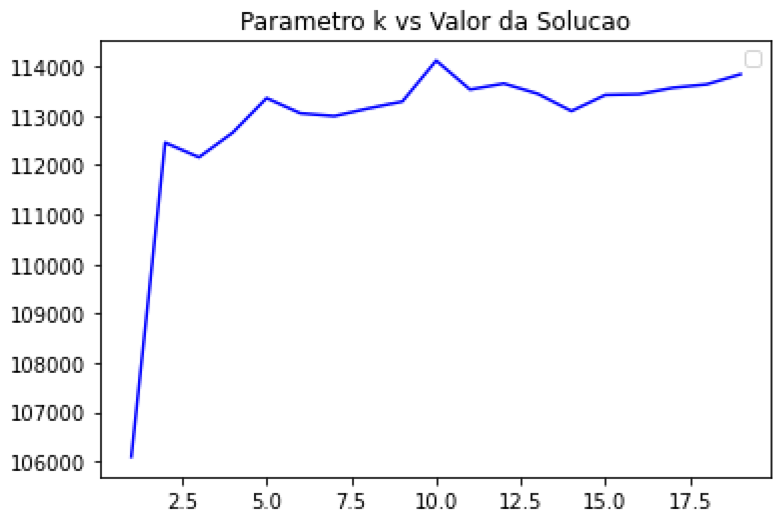

Figura 3: Variação do parâmetro k

Constata-se que a alteração do parâmetro k impacta muito significativamente na qualidade da solução em valores mais baixos, eventualmente tendendo a uma estabilização após certo crescimento.

8. Resultados computacionais

inst	*SI*	*SF*	*100*(SI-SF)/SI*	*desvioOtimalidade*	*tempoSA*	*tempoSolver*
1	104653	114099	-9,02601932	70,690412	10,8708s	22,51332s
2	104653	113923	-8,85784449	82,441373	10,8904s	22,79364s
3	104653	113828	-8,76706831	91,228007	10,6314s	23,53586s
4	327932	333694	-1,75707158	70,799139	11,6382s	22,46233s
5	327932	333732	-1,76865935	99,982515	12,0141s	22,67369s
6	327932	333547	-1,71224522	91,243600	12,2032s	22,60373s
7	417279	433669	-3,92782766	78,565360	46,5468s	22,29812s
8	417279	437535	-4,85430611	83,129684	45,6407s	23,79311s
9	417279	437916	-4,94561192	91,559425	48,4573s	24,23269s
10	1302137	1318988	-1,294103462	70,692594	51,0473s	23,15836s
11	1302137	1316608	-1,111326995	82,447286	52,1454s	23,39162s
12	1302137	1317383	-1,170844542	91,223643	52,9194s	22,36284s

Em que *inst* representa o número da instância, *SI* o valor da solução inicial encontrada pelo algoritmo guloso; *SF* o valor solução final encontrada pelo algoritmo de simulated annealing, *desvioOtimalidade* o valor do desvio percentual de *SF* em relação à solução ótima encontrada via solver e *tempoSA* e *tempoSolver* o tempo de execução do algoritmo de simulated annealing e do solver, respectivamente.

9. Conclusões

Os resultados obtidos indicam que a implementação do algoritmo não performou próximo à otimalidade, mas resultou em melhora do algoritmo guloso em todos os casos. A nossa configuração proposta para a execução do algoritmo é: Tmin = 1, cooling_rate = 0.87, k = 10.