Análise das técnicas de Têmpera Simulada e Algoritmos Genéticos na busca de soluções para o Problema do Caixeiro Viajante

Gustavo F. Armênio[1], Daniel A. P. de Castro[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR) - Curitiba - PR - Brasil

Resumo. *Este documento descreve a modelagem computacional do problema clássico de otimização combinatória "Caixeiro Viajante", objetivando a implementação das técnicas de Têmpera Simulada e busca local AG. Uma análise é feita a partir disso, descrevendo definições formais, metodologias, resultados e conclusões críticas de tais componentes. Assim, avaliam-se os procedimentos adotados pelos membros da equipe para encontrar soluções a partir de escolhas de modelagem e avaliação dos resultados.*

1. Problema do Caixeiro Viajante

Sendo um problema clássico de otimização na combinatória, o problema do caixeiro viajante representa um contexto em que um vendedor deve visitar n cidades em uma ordem que minimize a distância percorrida, de forma que visite uma única vez cada cidade e termine na mesma cidade em que partiu.

Canonicamente, o problema é representado como um grafo completo, cujos vértices representam as cidades e arestas as distâncias de cada cidade a outra. Ou seja, cada par de vértices (i, j) tem um custo associado $c(i, j)$ de visitar j a partir de i ou vice-versa, e a distância total é a soma dos custos c tomando cada par de vértices adjacentes no caminho de uma solução. Uma solução para o problema, portanto, é um caminho cíclico que contenha todos os vértices, no qual todos os vértices possuam exatamente duas arestas.

[Cormen et al. 2012] define o problema de decisão correspondente ao problema do caixeiro viajante como:

$$\begin{aligned} PCV = \{\ (G, c, k) : G = (V, E) \text{ é um grafo completo,} \\ \text{c é uma função de } V \times V \to Z, \\ k \in Z, \text{ e} \\ \text{G tem uma solução de custo máximo igual a } k\}\ \ (1) \end{aligned}$$

1.1. Representação das soluções

Dado que S é o espaço que contém todas as soluções possíveis, $P = u_0, u_1, ..., u_n$ é uma permutação dos n nós, u_i é um nó visitado e i se refere à ordem de visitação do nó, uma solução $s \in S$ é uma sequência de nós e é definida como:

$$s = u_0, u_1, ..., u_n, u_{n+1}, \text{ tal que } u_{n+1} = u_0 \quad (2)$$

A adição do primeiro nó visitado ao final da sequência é necessária para garantir um caminho fechado, ou seja, que o vendedor esteja de volta na primeira cidade visitada ao final do problema.

Dada a definição prévia, vamos modelar $D(s)$, a função que determina o custo total de uma solução s:

$$D(s) = \sum_{i=0}^{n} c(u_i, u_{i+1}) \tag{3}$$

Segundo a definição (1), $\forall s \in S$, $D(s) \leq k$, dado que $\exists s \in S$ tal que $D(s) = k$.

2. Metodologia

A implementação do problema e das abordagens Têmpera Simulada e Algoritmos Genéticos se baseou nos materiais de aula [Tacla 2024b] [Tacla 2024a], em seus formatos canônicos, e foi realizada em linguagem Python (disponível em `https://github.com/gustavo-fardo/caixeiro-viajante-SI/`). As escolhas de modelagem e implementação são detalhadas nesta seção.

2.1. Modelagem e implementação dos grafos-problema

A modelagem do problema do caixeiro viajante em grafos se deve à capacidade de grafos representarem relações de conectividade e distância – ou custo – entre dados elementos de um problema. No contexto desse problema, torna-se uma decisão sensata escolher uma representação que facilite o acesso rápido a estes valores, como as Matrizes de Adjacências.

Novos arranjos de cidades podem facilmente ser definidos a partir dos custos de cada nó a outro e a leitura do custo entre dois nós pode ser realizada com uma operação de acesso aos índices i e j (ou vice-versa) da matriz. Para a realização de testes, um grafo é criado aleatorizando $\frac{n \times (n-1)}{2}$ valores inteiros – ou seja, metade do número de arestas menos a diagonal principal, já que a matriz é simétrica – de 1 a um valor limite M, preenchendo também a diagonal principal com zeros (o M escolhido foi 10, o que limita as possibilidades de custos).

2.2. Modelagem e implementação das soluções

Baseando-se na definição formal das soluções dada pela equação (2), a sua implementação foi realizada com um vetor, formado por uma permutação dos índices de cada nó na matriz concatenado ao final com o primeiro valor da permutação gerada.

A análise de custo de uma solução se tornou, portanto, a tarefa de percorrer o vetor de solução, somando o custo das duplas entre um índice e seu próximo no vetor, consultando a matriz de adjacências, como modelado na equação (3).

2.3. Têmpera Simulada

Na implementação da têmpera, a escolha da equação de decaimento da temperatura T é essencial para estabelecer o comportamento das variações estocásticas ao longo da execução; a escolha feita foi pela exponencial decrescente, como visto em aula, que dita

um decaimento com taxa acelerada conforme o tempo passa, de forma que ao final do período haja pouca probabilidade de se realizar pulos para soluções piores. Considerando a temperatura inicial 100, a temperatura final 0, um dado número i de iterações e uma iteração número t, $T - schedule$ é a seguinte equação:

$$T - schedule(t) = 100 \times (\frac{i - t}{i})^2 \tag{4}$$

A única escolha realizada além da equação de $T - schedule$ se refere ao valor máximo de iterações i, devido à escolha de modelagem do decaimento que depende da razão de dado contador t e i. Ademais, a implementação seguiu a definição canônica vista em aula, explicitada no Algoritmo 1.

Algorithm 1 Têmpera Simulada ($solucao\_atual$, $n\_iteracoes$)

for $t \leftarrow 0$ to $i\_iteracoes$ **do**
  $T \leftarrow T - schedule(t)$
  if $T = 0$ **then return** $solucao\_atual$
  end if
  $solucao\_prox \leftarrow$ nova solucao gerada segundo a descrição da seção 2.2
  $\Delta E \leftarrow D(solucao\_prox) - D(solucao\_atual)$ ▷ Custos segundo a equação (3)
  if $\Delta E < 0$ **then**
    $solucao\_atual \leftarrow solucao\_prox$
  else
    $solucao\_atual \leftarrow solucao\_prox$ com probabilidade $e^{\frac{T}{\Delta E}}$
  end if
end for

2.4. Algoritmo Genético

A estratégia de escolha do modelo de AG envolveu diversos fatores fundamentados em suas etapas de implementação. De início, é definida uma população (de tamanho N) de soluções as quais a cada geração são avaliadas por uma função *fitness* inversamente proporcional ao custo (já que uma melhor solução é a de menor custo) para possibilitar a melhor seleção da roleta em seus intervalos de probabilidade. Após selecionados, os cromossomos se reproduzem a fim de gerar filhos de acordo com o parâmetro *p_cross* e mutar de acordo com os parâmetros *p_mut* e *i_mut* (escolha de mutação por posição, ordem ou embaralhamento). Por fim, são selecionados os N melhores dentre os pais e filhos mutados, perpetuando novos cromossomos para a próxima iteração de geração. O Algoritmo 2 apresenta o pseudo-código.

Durante a modelagem, percebeu-se o impacto de cada um dos parâmetros na diminuição custo ao longo das gerações. Já que o exemplo do problema utiliza populações de indivíduos com poucos genes, foi escolhida a mutação por embaralhamento com um valor pequeno (0.05, como sugerido no material) de p_cross para estabelecer uma mudança sucinta a cada nova geração. Outro detalhe que vale mencionar é a quantidade de indivíduos: optou-se por uma população equilibradamente maior ($N = 12$) a ponto de variar os valores dos custos. Assim, a roleta pôde estabelecer uma elitização maior (por meio do *fitness*) para os cromossomos favoráveis. Por último, o parâmetro de p_cross

provou-se ser relevante ao determinar um valor de considerável probabilidade (como o sugerido de 0.8) para atribuir chances de reprodução entre cromossomos.

Algorithm 2 Algoritmo Genético (N, max_ger, p_cross, p_mut, i_mut)

$populacao \leftarrow N$ soluções atribuídas inicialmente
for $t \leftarrow 0$ to max_ger **do**
    $pais \leftarrow populacao$
    $custos \leftarrow calcula\_custos(pais)$
    $fitness \leftarrow fitness(custos)$
    $selecionados \leftarrow roleta(pais, fitness)$
    $cruzados \leftarrow cruzamento(selecionados, p\_cross)$
    $mutados \leftarrow mutacao(cruzados, p\_mut, i\_mut)$
    $populacao \leftarrow seleciona\_N\_melhores(pais, mutados)$
end for

3. Resultados

A geração de gráficos que representam a evolução do custo em relação ao tempo para três execuções de cada algoritmo foi obtida com a biblioteca *matplotlib* do *python*. Esses gráficos estão dispostos nos Apêndices A, B e C, na seção de Apêndice.

As modelagens para os grafos em Matriz de Adjacências e as soluções foram efetivas para a representação. Quanto à funcionalidade, a escolha de sucessores apresentou um impacto principalmente na Têmpera Simulada, por aleatorizar uma permutação nova, nada relacionada com a solução anterior.

Na elaboração da Têmpera Simulada, o mais desafiador foi a escolha de $T - schedule$, sendo a descida exponencial efetiva para evitar pulos nas últimas iterações. Podemos ver nos Apêndices A e B (10 cidades e, respectivamente, 100 e 1000 iterações) que este algoritmo apresenta uma alta variação de custos na maior parte do tempo, exceto ao final em que tende a estabilizar. Ele tende a encontrar soluções razoáveis em pouco tempo, muitas vezes finalizando em uma solução pior que alguma das anteriores. Ademais, não apresenta um benefício tão relevante com o aumento do número de iterações, devido à alta probabilidade de pulos a soluções piores durante maior parte das iterações, reflexo da escolha do método de geração de sucessores.

Já para a implementação do Algoritmo Genético, houve um cuidado especial em determinar a função *fitness* adequada ao problema em questão. Para tanto, utilizou-se o complemento de cada valor de custo sobre o total, o que tornou a função inversamente proporcional a ele. Além disso, uma dificuldade encontrada ao longo da modelagem foi definir valores equilibrados para os diversos parâmetros que o AG possui, já que foi possível constatar uma quantificação de etapas relativamente maior em comparação a outros algoritmos de busca anteriormente estudados.

Como é possível ver nos Apêndices A e B, o Algoritmo Genético tem uma tendência majoritaria de descida no custo, ou seja, ao longo das gerações a população tende a se uniformizar. Isso se deve à quantidade reduzida de mutações, escolhida a fim de evitar um comportamento mais estocástico como o da Têmpera Simulada.

No Apêndice C, é evidente perceber o impacto do aumento do número de cidades (de 10 para 40) para o Algortimo Genético, e a importância de um número de gerações proporcional à complexidade do problema. A quantidade de 1000 gerações se mostrou deficiente para 40 cidades, visto que o AG demonstrou maior estabilidade muito próximo ao final das iterações. Por outro lado, a Têmpera apresentou um comportamento muito semelhante dos casos dos Apêndices A e B, com alta variabilidade, exceto no final, e resultados razoáveis.

4. Comparações entre os métodos de busca

Como visto na definição de Algoritmos de Busca Local, as ações de cada solução não são relevantes, e sim o conjunto de soluções que vão se aprimorando a ponto de se aproximar da escolha ótima para o problema. Por essa razão a técnica é definida como metaheurística, já que a informação específica do problema não é previamente implementada, e sim adquirida ao longo das iterações, uma importante difereça na implementação em relação a outros tipos de Busca.

A principal vantagem da Busca Local com AG e Têmpera Simulada é a sua capacidade de selecionar soluções piores que as encontradas anteriormente a fim de evitar mínimos locais e encontrar soluções ótimas. Já as buscas cegas e heurísticas podem apresentar uma tendência a estabilizar em mínimos locais. Contudo, em termos de modelagem, AG e Têmpera Simulada apresentam uma dependência alta da calibração de determinados parâmetros para uma funcionalidade de maior performance, enquanto as buscas cegas e heurísticas têm um comportamento bem-definido e independente da relação problema-parâmetros – com uma ressalva para a escolha da função heurística para as buscas informadas, cuja efetividade pode mudar para diferentes problemas.

Comparando entre as Buscas Locais, o Algoritmo Genético e a Têmpera Simulada, temos que AG consistentemente encontra soluções iguais ou melhores que Têmpera, se dado gerações suficientes. Com um número limitado de iterações (ou gerações), a Têmpera Simulada pode encontrar soluções razoáveis em pouco tempo; contudo, não consegue tirar grande proveito do aumento de iterações dada a descida exponencial escolhida e comumente não finaliza na melhor solução encontrada em toda a busca.

5. Considerações finais

A distribuição do trabalho entre os membros contou com uma ênfase do aluno Gustavo na modelagem e implementação dos grafos, soluções e da Têmpera Simulada e do aluno Daniel no Algoritmo Genético, que resultaram em esforços de complexidade semelhante para cada. O desenvolvimento conjunto e colaborativo proporcionou resoluções mais rápidas de dúvidas e erros de implementação e o processo de escrita foi realizado em comunhão de forma a dispor as informações de forma mais coesa e clara. O gasto de horas estimado ao longo do processo foi em torno de 10 horas por membro.

Referências

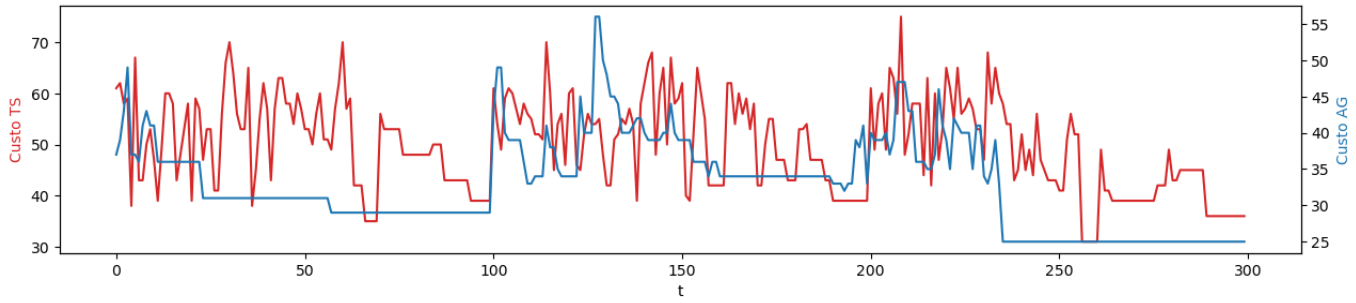
Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2012). *Algoritmos*. Elsevier, Rio de Janeiro.

Tacla, C. A. (2024a). *Algoritmos Genéticos*. UTFPR, Curitiba.

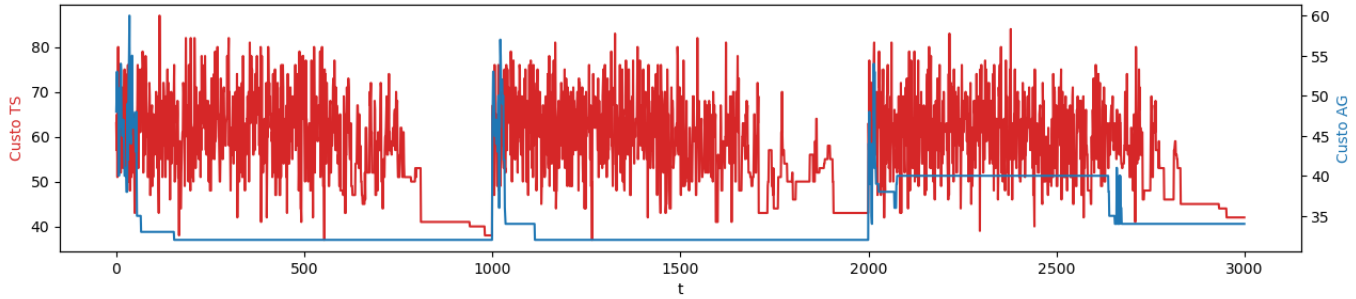
Tacla, C. A. (2024b). *Busca Local*. UTFPR, Curitiba.

Apêndice

A. (Custos x t) 3 execuções para 100 iterações (ou gerações) e 10 cidades. Em Vermelho, Têmpera Simulada e em Azul, Algoritmo Genético.



B. (Custos x t) 3 execuções para 1000 iterações (ou gerações) e 10 cidades. Em Vermelho, Têmpera Simulada e em Azul, Algoritmo Genético.



C. (Custos x t) 3 execuções para 1000 iterações (ou gerações) e 40 cidades. Em Vermelho, Têmpera Simulada e em Azul, Algoritmo Genético.