Resolução do Problema do Caixeiro Viajante Utilizando *Simulated Annealing*

Victor Raposo Ravaglia de Oliveira

1 Introdução

Esse trabalho tem como objetivo avaliar o desempenho do algoritmo de *Simulated Annealing* [1](SA) para a solução de uma configuração Problema do Caixeiro Viajante [1](PCV). Mais especificamente, pretende-se avaliar o uso de uma heurística específica para um diferente número de cidades. Nesse estudo, as cidades estão dispostas ao longo de um percurso em forma de uma estrela de 5 pontas. Uma vantagem dessa configuração é que a distância mínima a ser percorrida é sempre conhecida.

O documento está dividido da maneira que se segue. Na Seção 2 é apresentado o embasamento teórico necessário para entender a heurística avaliada e a configuração do PCV. A Seção 3 descreve a metodologia e a implementação utilizados para obter os resultados. Na Seção 4 resultados de execuções do SA para diferentes números de cidades são apresentados. Finalmente, a Seção 5 apresenta conclusões sobre o trabalho.

2 Teoria

O *Simulated Annealing* (SA) é um algoritmo que pode ser utilizado para diversos problemas de otimização [2]. Os parâmetros do SA variam de acordo com a função que deseja-se otimizar. Além disso, heurísticas podem ser implementadas na expectativa melhorar o desempenho do algoritmo [3]. Nesse trabalho, uma configuração específica do Problema do Caixeiro Viajante (PCV) e uma heurística de atualização de estados são estudadas.

2.1 Configuração do PCV

O PCV consiste em, para uma dada lista de cidades e as distâncias entre elas, encontrar o menor caminho necessário para percorrer todas elas. O PCV pode ser definido com diferentes configurações, que definem regras que devem ser seguidas ao definir a ordem em que as cidades são percorridas (*e.g.* se cidades podem ser visitadas mais de uma vez, se a primeira cidade é sempre a mesma, etc).

Para configuração do PCV estudada, todas as cidades são posicionadas ao longo de uma circunferência de raio variável. A posição das cidades são coordenadas cartesianas de duas dimensões. As cidades devem ser percorridas exatamente uma vez e o trajeto deve terminar na primeira cidade escolhida, seja ela qual for. O raio da circunferência varia conforme uma onda triangular de frequência 10π, de modo que o posicionamento resultante se assemelhe a uma estrela. A Figura 1 ilustra a configuração do PCV utilizada para 100 cidades. O circulo tem raio 0.85 e a onda triangular amplitude 0.2 e valor médio 0.2, todos valores arbitrários. Para essa configuração de cidades, o mínimo global é sempre o caminho que envolve percorrer as cidades em ordem crescente ou decrescente.

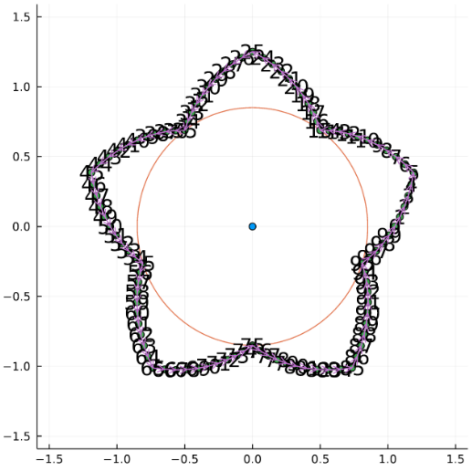


Figura 1: Ilustração da configuração PCV utilizada para os testes do trabalho para 100 cidades.

2.2 SA e a Heurística Estudada

O SA é um método de otimização com o seguinte algoritmo:

- Passo 1: Inicializar x_i um valor aleatório dentro do espaço de soluções Inicializar temperatura $T = T_0$, o contador de iteração $k = 1$ e o contador N
- Passo 2: Calcular $J(x_i)$
- Passo 3: Selecione $\hat{x_i} = P(x_i)$
- Passo 4: Calcular $J(\hat{x_i})$
- Passo 5: Calcular $\Delta J = J(x_i) - J(\hat{x_i})$
  - Calcular $p = \exp(-\Delta J/T)$
  - se $J(\hat{x_i}) < J(x_i)$
    - $x_i = \hat{x_i}$

[1]https://en.wikipedia.org/wiki/Travelling_salesman_problem

  - senão se $[rand[0,1) < p]$
    * $x_i = \hat{x}_i$
  - se $J(x_i) < J_{min}$
    * $J_{min} = J(x_i)$
    * $x_{min} = x_i$
  - se $J_{min} == J_{otimo}$
    * Vá para o passo 8
  - Atribuir $n = n + 1$
- Passo 6: Se $n > N$
  - Atribuir $T = T_0/\log 2(1 + k)$
  - Atribuir $n = 0$
- Passo 7: Se $k < K_{max}$ Vá para o passo 3
- Passo 8: Retorne a solução J_{min}

A partir de um estado atual x_i, o SA gera perturbações $P(x_i)$ gerando um estado candidato $\hat{x_i}$ para a próxima iteração do algoritmo. O objetivo do SA nesse caso, é minimizar uma função custo $J(x)$. No caso do PCV, x_i representa uma possível ordem em que as cidades podem ser percorridas e $J(x_i)$ a distância percorrida ao seguir a ordem x_i. O estado candidato $\hat{x_i}$, é o próprio estado x mas com uma permutação no índice de duas cidades aleatórias. Caso $\hat{x}$ seja aceito como próximo estado, ele é comparado com estado de menor custo encontrado até o momento x_{min}. O algoritmo finaliza sua execução caso $k > K_{max}$ ou caso encontre o custo mínimo global J_{otimo}.

Uma possível heurística que pode ser implementada para esse problema é, a cada incremento do valor de k, alterar o valor do estado atual para que $x_i = x_{min}$. Ao longo desse documento, essa heurística será referenciada apenas como "a heurística".

3 Metodologia

Para avaliar o desempenho da heurística, são realizadas execuções do SA para diferentes quantidades de cidades. Iniciando com 10 cidades, uma grade de parâmetros é testada para 10 diferentes sementes, tanto com quanto sem utilizar a heurística. Em seguida, a grade é reduzida baseando-se no desempenho obtido. A grade reduzida é então utilizada para os demais números de cidades. Caso o algoritmo obtenha sucesso em pelo menos uma configuração em pelo menos uma semente, o número de cidades é aumentado. Caso contrário, os parâmetros da grade são ajustados.

3.1 Ajuste da grade de parâmetros

A partir de testes inicias com número relativamente pequeno de cidades (10) são estabelecidas as seguintes faixas de valores iniciais para os parâmetros do SA: $N = 10^4$, $K_{max} = 50$, $T_0 = [0.1,\ 0.2,\ \ldots,\ 1.0]$. Essas faixas de valores correspondem a grade reduzida. Em seguida, 10 sementes são testadas, caso o ótimo global não seja encontrado, N ou K_{max} é aumentado. A grade de parâmetros é a mesma para execuções com ou sem o uso da heurística.

C	N	K_{max}	T_0
10	10^4	50	[0.1, 1.0]
20	10^4	50	[0.1, 1.0]
40	10^4	50	[0.1, 1.0]
60	10^4	50	[0.1, 1.0]
80	10^4	50	[0.1, 0.5]
100	10^4, 10^5	50, 100	[0.1, 0.5]
150	10^5	100	[0.1, 0.5]
200	10^5	100, 250, 500	[0.1, 0.5]
250	10^5	500, 1000, 2000	[0.1, 0.5]

Tabela 1: Histórico de ajustes dos parâmetros do SA para cada número de cidades (C). As temperaturas T_0 indicam intervalos com incremento de 0.1 (*e.g.* 0.1, 0.2, ... 0.5).

A Tabela 1 apresenta as mudanças realizadas no parâmetros para cada aumento no número de cidades. Os melhores resultados se concentram na faixa de $T_0 = [0.1,\ 0.2,\ \ldots,\ 0.5]$, por isso foi modificada a partir de 80 cidades. O valor de k em que o algoritmo obtém seu menor custo é salvo no final de cada execução. Caso k seja próximo de K_{max}, o valor de K_{max} é aumentado. Valores de N maiores que 10^5 não apresentam melhora significativa no desempenho do algoritmo e aumentam consideravelmente o tempo de execução (o número de iterações do SA é proporcional a N).

Para cada execução do algoritmo, caso o mesmo encontre o ótimo global, a execução é interrompida. Consequentemente, execuções diferentes com os mesmos valores de K_{max} e N, e que o ótimo global não é encontrado, tem tempos de execução semelhantes. Analogamente, execuções em que o mínimo global é encontrado, têm tempos de execução diferentes pois o algoritmo é interrompido prematuramente.

3.2 Implementação

A linguagem Julia [2] foi utilizada para realizar todas as execuções do SA. Para reduzir o tempo de execução do programa, houve um foco em reduzir ao máximo o número de alocações em memória. Isso foi feito declarando todos os possíveis parâmetros como constantes e pré-alocando os vetores de histórico de temperaturas e custos. Além disso, os vetores de estado x_i e $\hat{x_i}$ tiveram apenas os valores de cada índice alterado a cada iteração do programa. A alternativa para isso seria criar uma cópia simples dos vetores, o que acarretaria em alocar um novo espaço de memória a cada iteração do SA.

Outra medida de otimização adotada, foi calcular uma matriz de distâncias euclidianas X_d tal que:

$$X_d = \begin{bmatrix} ||x_1 - x_1|| & ||x_1 - x_2|| & \ldots & ||x_1 - x_c|| \\ ||x_2 - x_1|| & ||x_2 - x_2|| & \ldots & ||x_2 - x_c|| \\ \vdots & \vdots & \ddots & \vdots \\ ||x_c - x_1|| & ||x_c - x_2|| & \ldots & ||x_c - x_c|| \end{bmatrix},$$

[2] https://julialang.org/

onde c é o número de cidades e x_n a posição euclidiana da cidade de índice n. Assim, ao calcular a função custo, a matriz X_d é lida conforme a sequência de cidades presente no trajeto x_i ou $\hat{x_i}$. Consequentemente, não é necessário calcular novamente as distâncias a cada iteração do SA. O número total de execuções E do SA, para cada número de cidades é dado por:

$$E = T_{0s} \times N_s \times K_{maxs} \times seeds \times 2,$$

onde T_{0s}, N_s, K_{maxs} e $seeds$ são o número de possibilidades de T_0, N, K_{max} e sementes para cada execução, respectivamente. O fator 2 é devido ao fato de que o algoritmo é executado tanto com quanto sem o uso da heurística.

A cada execução, o histórico de temperaturas e custos foi guardado para que fosse possível analisar o comportamento da curva de custos. Para cada execução, o custo mínimo encontrado e os parâmetros utilizados foram salvos em arquivo. Os parâmetros em conjunto com a semente utilizada permitem reproduzir os resultados caso necessário. Ao final de todas as execuções, para um dado número de cidades e uso ou não da heurística, a execução que foi capaz de obter o menor custo (J_{min}) é considerada a melhor execução e é salva em outro arquivo. Caso mais de uma execução seja capaz de encontrar o mínimo global, o tempo de execução é utilizado como critério de desempate (quanto menor melhor).

4 Resultados

Seguindo a metodologia descrita na Seção 3, foram obtidos resultados para diferentes números de cidades, comparando o uso ou não da heurística. Além do custo mínimo encontrado, é observado o tempo necessário e os parâmetros que levam a uma execução particular do SA chegar a um dado resultado.

A Figura 2 apresenta uma comparação geral entre o uso ou não da heurística. Para cada iteração do SA, é mostrado o tempo médio e o menor custo encontrado, ambos normalizados com relação ao número de cidades. É possível ver que o desempenho tanto com relação ao tempo quanto ao menor custo é semelhante.

A Tabela 2 apresenta os resultados médios obtidos para cada execução do SA. Pode-se observar que, os resultados serem semelhantes entre o uso ou não da heurística para C < 150. No entanto, para os parâmetros testados, apenas a configuração sem heurística foi capaz de encontrar resultados ótimos para um número de cidades C > 100.

A Tabela 3 apresenta os parâmetros e resultados obtidos para as melhores execuções do SA, para cada número de cidades. É possível ver que, para os parâmetros testados, o mínimo global foi encontrado apenas em casos que a heurística não foi utilizada.

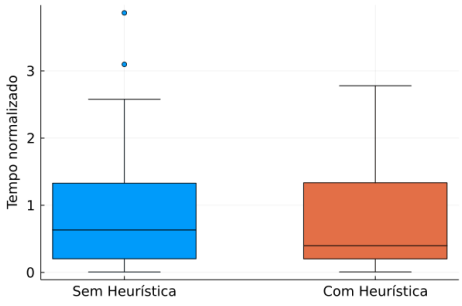
A Figura 3 apresenta uma comparação entre os custos obtidos a cada iteração do SA para 150 cidades. Esse é menor quantidade de cidades testada em que o uso da heurística não resultou em um ótimo global. Na Figura 3b é possível ver que o algoritmo não é capaz de convergir, apesar de ter um comportamento semelhante a Figura 3a.

C	TE		% Ótimo		MT (s)	
	SH	CH	SH	CH	SH	CH
10	100	100	100%	100%	0.01	0.01
20	100	100	100%	100%	0.01	0.01
40	100	100	74%	74%	0.03	0.03
60	100	100	33.5%	31%	0.21	0.19
80	50	50	11.5%	10.5%	0.28	0.29
100	200	200	5.0%	2.5%	0.56	0.59
150	50	50	1.0%	0%	1.47	1.48
200	150	150	1.3%	0%	7.06	7.28
250	150	150	0.5%	0%	33.4	35.6

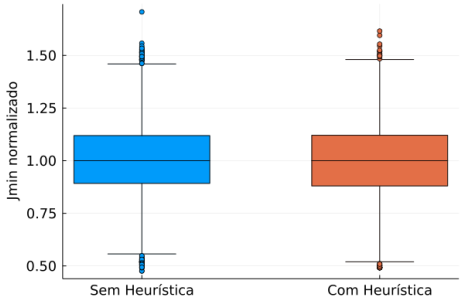
Tabela 2: Total de execuções (TE) do SA, percentual de vezes em que o ótimo global foi encontrado (% Ótimo) e média de tempo (MT) de cada execução, para cada número de cidades (C). "SH" e "CH" representam execuções com e sem uso da heurística, respectivamente.

C	T_0		J_{min}		Tempo (s)	
	SH	CH	SH	CH	SH	CH
10	0.1	0.5	**6.49**	**6.49**	6e-5	6e-5
20	0.2	0.2	**7.68**	**7.68**	0.0001	0.0002
40	0.1	0.1	**7.71**	**7.71**	0.0015	0.0015
60	0.1	0.3	**7.72**	**7.72**	0.0045	0.0081
80	0.2	0.3	**7.72**	**7.72**	0.013	0.033
100	0.2	0.3	**7.72**	**7.72**	0.23	0.66
150	0.2	0.3	**7.65**	11.1	1.94	1.96
200	0.2	0.2	**7.72**	7.80	8.67	12.78
250	0.2	0.2	**7.68**	14.24	58.29	59.1

Tabela 3: Melhores execuções do SA e seus respectivos parâmetros. "SH" e "CH" representam execuções com e sem uso da heurística, respectivamente. Os valores de J_{min} em negrito indicam ótimos globais.

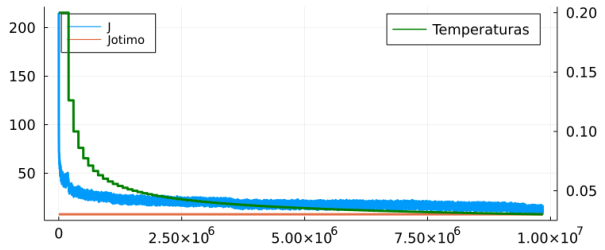


(a) Comparação de tempos.

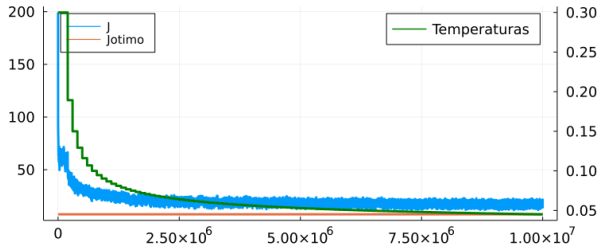


(b) Comparação de custos mínimos.

Figura 2: Comparação do tempo de execução e custo mínimo obtido com e sem o uso da heurística. Os tempos e custos estão normalizados em relação a respectiva média para cada número de cidades.



(a) Sem utilizar a heurística.



(b) Utilizando a heurística.

Figura 3: Comparação das curvas de custo com e sem o uso da heurística para 150 cidades.

5 Conclusão

O algoritmo de *Simulated Annealing* foi avaliado minimizando o custo do Problema do Caixeiro Viajante. Uma grade de parâmetros foi estabelecida a partir de testes inicias com 10 cidades. Os parâmetros do algoritmo foram ajustados conforme o número de cidades foi incrementado. Para cada combinação de parâmetros, 10 sementes foram testadas em busca do menor custo. O mínimo global foi encontrado para diferentes números de cidades, até um valor máximo de 250. Para os parâmetros testados, a comparação entre o uso ou não da heurística que estabelece $x = x_{min}$ na troca de temperatura, se demonstrou inconclusiva até 100 cidades. Apesar disso, para um número de cidades maior que 100, o SA encontrou o ótimo global apenas em casos em que não utilizou a heurística.

Referências

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598): 671–680, 1983.

[2] Nazmul Siddique and Hojjat Adeli. Simulated annealing, its variants and engineering applications. *International Journal on Artificial Intelligence Tools*, 25(06): 1630001, 2016.

[3] Alan R. McKendall, Jin Shang, and Saravanan Kuppusamy. Simulated annealing heuristics for the dynamic facility layout problem. *Computers & Operations Research*, 33(8):2431–2444, 2006. ISSN 0305-0548. doi:https://doi.org/10.1016/j.cor.2005.02.021.