centro universitário FEI

IAR
Inteligência Artificial Aplicada à Automação e Robótica

Programação Científica

Prof. Dr. Danilo H. Perico

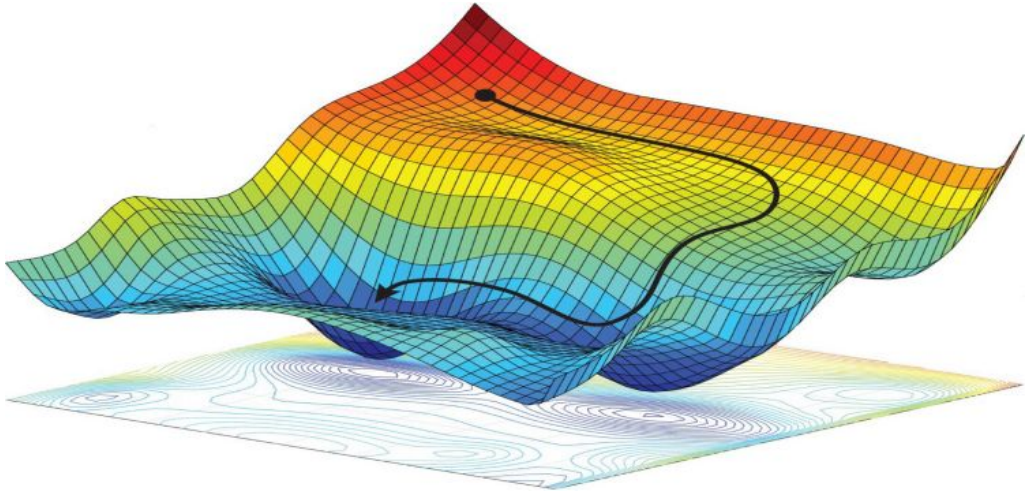
OTIMIZAÇÃO

OTIMIZAÇÃO

- Computação Numérica: uso de computadores para resolver problemas envolvendo números reais
    - Muitas soluções simples para problemas são computacionalmente inviáveis porque dependem de certos números específicos que não são computáveis
- **Otimização é o nome usual para um campo muito grande de pesquisa numérica**
- O objetivo da Otimização é normalmente encontrar a melhor solução para um problema que possui um grande número de soluções possíveis

OTIMIZAÇÃO

- O problema de **achar o valor máximo ou mínimo de uma função** objetivo, possivelmente sujeito a um conjunto de restrições, é conhecido como problema de otimização



OTIMIZAÇÃO

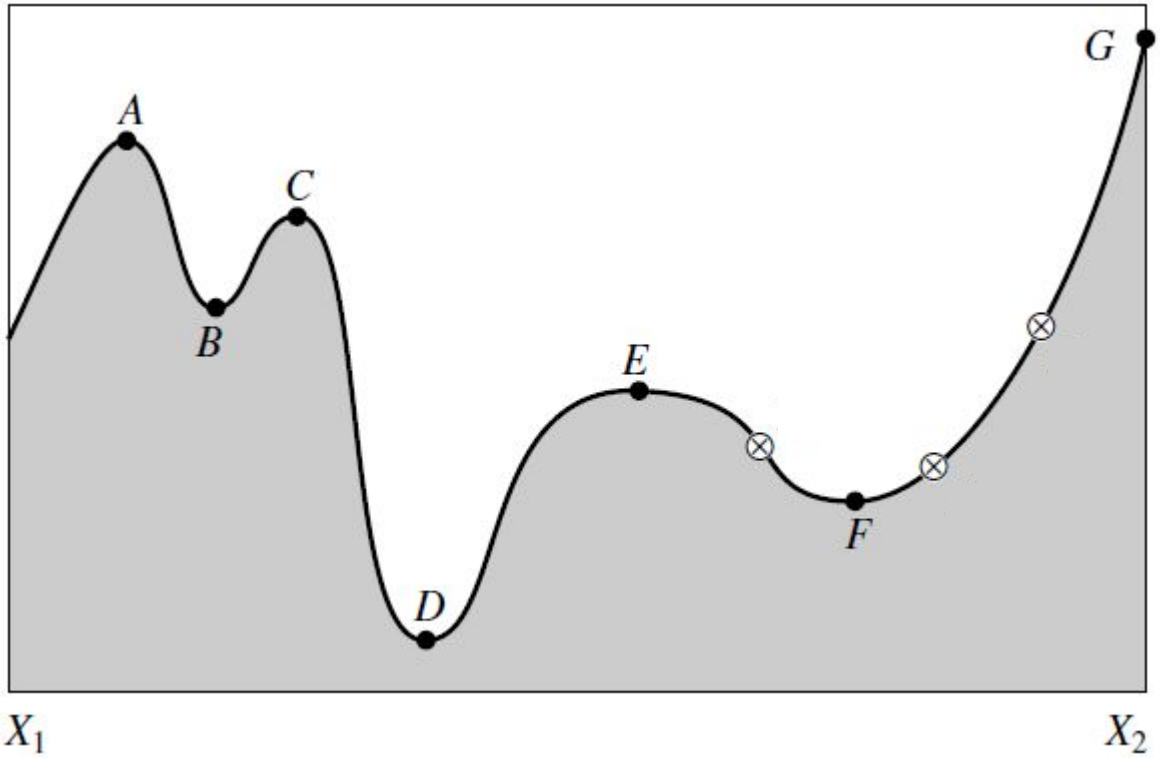
- Formalização:
  - **Dada uma função f que depende de uma ou mais variáveis independentes, encontre o valor dessas variáveis para que f assuma um valor máximo ou mínimo**
- As tarefas de maximização e minimização são trivialmente relacionadas entre si. Exemplo:
  - Minimizar $C = -2x-3y$ é o mesmo que maximizar $P = 2x+3y$, *pois* $P = -C$

OTIMIZAÇÃO

- Um extremo *(ponto máximo ou mínimo)* pode ser **global** ou **local**
  - ***Global:*** *verdadeiramente o valor mais alto ou mais baixo da função*
  - ***Local:*** *o mais alto ou mais baixo em uma vizinhança finita e não no limite dessa vizinhança*
- Encontrar um extremo global é, em geral, um problema muito difícil

OTIMIZAÇÃO

- Os pontos ***A***, ***C*** e ***E*** são máximos locais, mas não globais
- Os pontos ***B*** e ***F*** são mínimos locais, mas não globais
- O máximo global ocorre em ***G***
- O mínimo global está em ***D***



Fonte da imagem: Numerical Recipes

OTIMIZAÇÃO

- Na Otimização buscamos resolver um problema da forma mais rápida, com pouco uso de memória e com pouco custo computacional

OTIMIZAÇÃO

- Existem diversas estratégias para resolver o problema de otimização!
- Vamos estudar algumas:
  - **Programação Linear (*Linear Programming*)**
  - **Subida de Encosta (*Hill Climbing*)**
  - **Recozimento Simulado (*Simulated Annealing*)**
  - **Algoritmo Genético *(Genetic Algorithm)***

PROGRAMAÇÃO LINEAR

OTIMIZAÇÃO RESTRITA

- Na maior parte do tempo, estamos preocupados com **Otimização Restrita**

- Na Otimização Restrita, existem limitações *a priori* nos valores permitidos para as variáveis independentes

PROGRAMAÇÃO LINEAR

- **Programação Linear** é uma maneira bem estabelecida de **Otimização Restrita**

- Na Programação Linear, tanto a função a ser otimizada quanto as restrições são funções lineares das variáveis independentes

- Assim, a Programação Linear é uma família de problemas que busca otimizar uma **equação linear**:
  - Equação da forma $y = ax_1 + bx_2 + \ldots + c\square x\square$

PROGRAMAÇÃO LINEAR

- A Programação Linear terá os seguintes componentes:
  - Uma **função de custo** que queremos **minimizar**:
    - $c_1x_1 + c_2x_2 + \ldots + c_nx_n$
      - cada x é uma variável e está associada a um custo c

PROGRAMAÇÃO LINEAR

- A Programação Linear terá os seguintes componentes:
  - Uma **restrição**, que é representada como a soma de variáveis que é menor ou igual a um valor:
    - $a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$
  - ou precisamente igual a este valor:
    - $a_1x_1 + a_2x_2 + ... + a_nx_n = b$

PROGRAMAÇÃO LINEAR

- A Programação Linear terá os seguintes componentes:
  - **Limites** individuais em variáveis *(por exemplo, uma variável não pode ser negativa)* da forma:
    - $l_i \leq x_i \leq u_i$

PROGRAMAÇÃO LINEAR

- Quando a Programação Linear aparece em um formato particular onde todas as restrições são equações e todas as variáveis são não negativas diz-se que ela está na **forma padrão**

PROGRAMAÇÃO LINEAR

- Exemplo:
  - Considere **2 células de montagem: X_1** e **X_2**
    - **X_1** custa ***R$ 50 / hora*** para ser executada
    - **X_2** custa ***R$ 80 / hora*** para ser executada

PROGRAMAÇÃO LINEAR

- Exemplo:
  - **O objetivo é encontrar a quantidade de horas que cada linha precisa funcionar para produzir um total de peças e minimizar custos**
  - Isso pode ser formalizado como uma **função de custo**:
    - $50x_1 + 80x_2$

PROGRAMAÇÃO LINEAR

- Exemplo:
  - **X_1** precisa de **5** pessoas por hora para funcionar
  - **X_2** precisa de **2** pessoas por hora para funcionar
  - No total, tem-se **20** pessoas que podem ser usadas
  - Isso pode ser formalizado como uma **restrição**: $5x_1 + 2x_2 \leq 20$

PROGRAMAÇÃO LINEAR

- Exemplo:
  - X_1 produz **10** peças por hora
  - X_2 produz **12** peças por hora
  - A empresa precisa de **90** peças
    - Esta é outra **restrição**: $10x_1 + 12x_2 \geq 90$

PROGRAMAÇÃO LINEAR

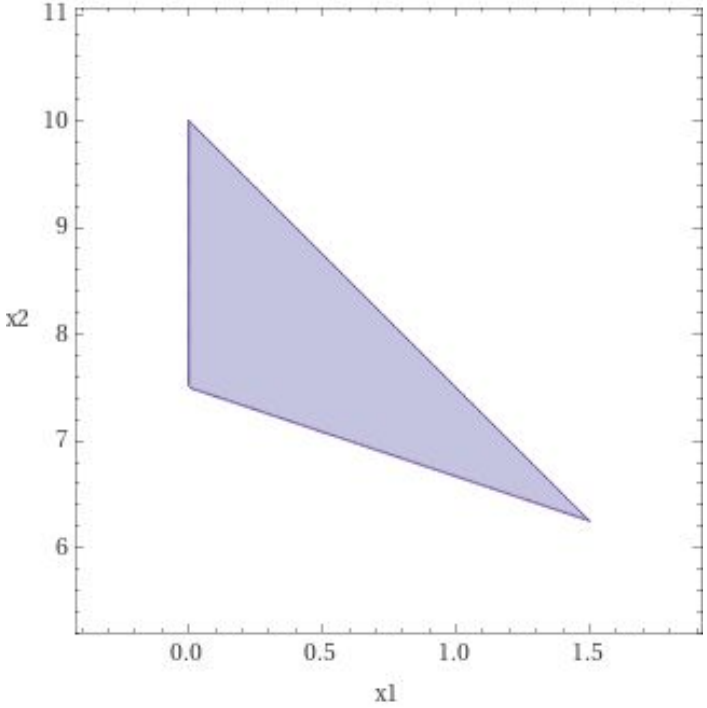
- Exemplo:
  - Problema: a restrição $10x_1 + 12x_2 \geq 90$ não pode ser assim
  - As restrições só podem aparecer nas formas:
    - $a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$ ou $a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$
  - Assim, multiplicamos por (-1):
    - $-10x_1 - 12x_2 \leq -90$

PROGRAMAÇÃO LINEAR

- Resumindo o problema apresentado:
- Minimizar a função de custo: $\xi = 50x_1 + 80x_2$
- Dadas as restrições:
  - $5x_1 + 2x_2 \leq 20$
  - $-10x_1 - 12x_2 \leq -90$

PROGRAMAÇÃO LINEAR

- Graficamente, a região demarcada encapsula todas as soluções viáveis, dadas as restrições



SIMPLEX

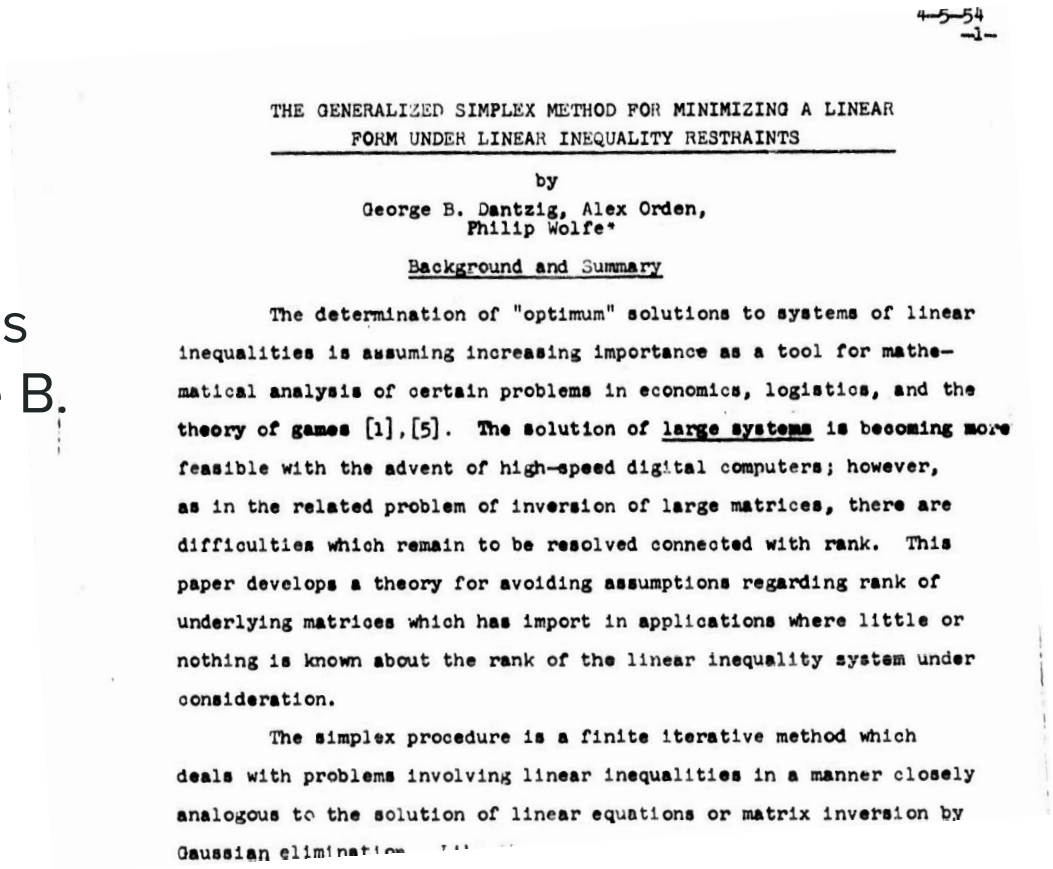
- O algoritmo ***Simplex*** é um dos mais conhecidos e utilizados para se resolver problemas relacionados à programação linear

- A ideia do ***Simplex*** é a seguinte:

    - Atribui-se valor zero às variáveis, o que seria distante da solução
    - Então, incrementa-se pouco a pouco a variável que tem maior interferência positiva no resultado da função objetivo, ou seja, a que possui o maior coeficiente

SIMPLEX

- Um dos primeiros artigos (1954) - Dantzig, George B.



4-5-54
-1-

THE GENERALIZED SIMPLEX METHOD FOR MINIMIZING A LINEAR FORM UNDER LINEAR INEQUALITY RESTRAINTS

by

George B. Dantzig, Alex Orden,
Philip Wolfe*

Background and Summary

The determination of "optimum" solutions to systems of linear inequalities is assuming increasing importance as a tool for mathematical analysis of certain problems in economics, logistics, and the theory of games [1],[5]. The solution of large systems is becoming more feasible with the advent of high-speed digital computers; however, as in the related problem of inversion of large matrices, there are difficulties which remain to be resolved connected with rank. This paper develops a theory for avoiding assumptions regarding rank of underlying matrices which has import in applications where little or nothing is known about the rank of the linear inequality system under consideration.

The simplex procedure is a finite iterative method which deals with problems involving linear inequalities in a manner closely analogous to the solution of linear equations or matrix inversion by Gaussian elimination

https://apps.dtic.mil/sti/pdfs/AD0114134.pdf

SIMPLEX

https://www.semanticscholar.org/paper/The-(Dantzig)-simplex-method-for-linear-programming-Nash/5a1abff37465f37f2d49c0ee5a610cf0cfb1c5ad

SIMPLEX

Linear Programming and the Simplex Method

David Gale

This exposition of linear programming and the simplex method is intended as a companion piece to the article in this issue on the life and work of George B. Dantzig in which the impact and significance of this particular achievement are described. It is now nearly sixty years since Dantzig's original discovery [3] opened up this whole new area of mathematics. The subject is now widely taught throughout the world at the level of an advanced undergraduate course. The pages to follow are an attempt at a capsule presentation of the material that might be covered in three or four lectures in such a course.

etc. Further, each food has a cost, so among all such adequate meals the problem is to find one that is least costly.

The Transportation Problem

A certain good, say steel, is available in given amounts at a set of m origins and is demanded in specified amounts at a set of n destinations. Corresponding to each origin i and destination j there is the cost of shipping one unit of steel from i to j. Find a shipping schedule that satisfies the given demands from the given supplies at minimum cost.

Almost all linear programming applications,

https://www.ams.org/notices/200703/fea-gale.pdf

PROGRAMAÇÃO LINEAR

- No Python, o ***Simplex*** pode ser utilizado pela biblioteca ***SciPy***:

PROGRAMAÇÃO LINEAR

- Exemplo em aula

PROGRAMAÇÃO LINEAR

- Exemplo 2: Digamos que uma fábrica produz quatro produtos diferentes e que a quantidade produzida diariamente do primeiro produto é x_1, a quantidade produzida do segundo produto é x_2 e assim por diante. O objetivo é determinar a quantidade de produção diária para maximizar o lucro de cada produto, levando em consideração as seguintes condições:

PROGRAMAÇÃO LINEAR

- O lucro por unidade de produto é de R$ 20, R$ 12, R$ 40 e R$ 25 para o primeiro, segundo, terceiro e quarto produto, respectivamente.
- Devido a restrições de mão de obra, o número total de unidades produzidas por dia não pode exceder cinquenta.
- Para cada unidade do primeiro produto, três unidades da matéria-prima A são consumidas.

PROGRAMAÇÃO LINEAR

- Cada unidade do segundo produto requer duas unidades da matéria-prima A e uma unidade da matéria-prima B. Cada unidade do terceiro produto precisa de uma unidade de A e duas unidades de B. Finalmente, cada unidade do quarto produto requer três unidades de B.

- Devido às restrições de transporte e armazenamento, a fábrica pode consumir até cem unidades da matéria-prima A e noventa unidades de B por dia

SUBIDA DE ENCOSTA
HILL CLIMBING

SUBIDA DE ENCOSTA

- **Tenta encontrar a melhor solução para um problema**
- Ideia geral:
  - Começa com uma solução aleatória
  - Gera vizinhos: soluções que diferem ligeiramente da atual
  - Se o melhor desses vizinhos for melhor do que o atual, ele substitui a solução atual por esta solução melhor

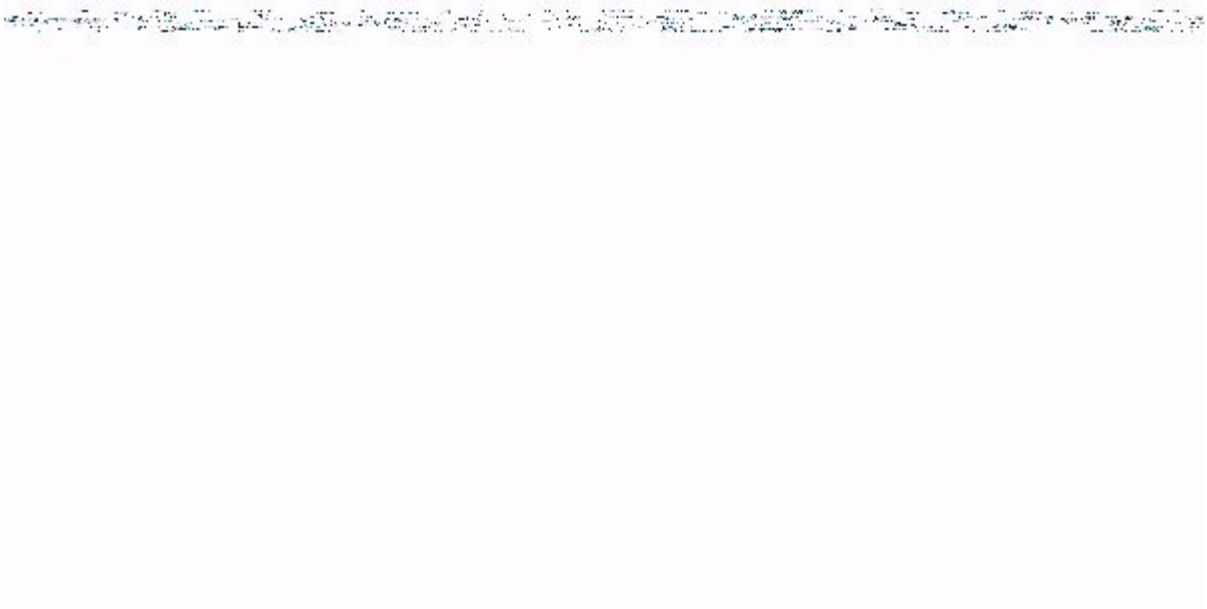
SUBIDA DE ENCOSTA

- Ideia geral:
  - Em seguida, ele repete o padrão criando novamente vizinhos
  - Se em algum ponto nenhum vizinho for melhor do que a solução atual, ele retorna a solução

SUBIDA DE ENCOSTA

- Como ele sempre tenta encontrar um vizinho melhor, é bem provável que não encontre a melhor solução **(máximo / mínimo global)**
- Mas é muito bom para encontrar um **máximo / mínimo local**!
    - Que, no fim, são soluções boas, mas não a melhor

SUBIDA DE ENCOSTA



Fonte da imagem: https://medium.com/data-hackers/algoritmos-de-otimiza%C3%A7%C3%A3o-hill-climbing-e-simulated-annealing-3803061f66f0

SUBIDA DE ENCOSTA

- Exemplo tradicional:
  - Suponha um jogo em que você precisa descobrir uma frase com tamanho conhecido: 13 caracteres
  - Como resolver esse problema? Podemos usar força bruta?
    - Tentar todas as possibilidades!
    - Vamos ver!

SUBIDA DE ENCOSTA

- Na tabela ASCII, temos ***95 caracteres*** que podem ser impressos
- Como seria a abordagem força-bruta? Testamos todas as possibilidades de caracteres combinados com todos caracteres!
  - **13 estruturas de repetição, cada uma como 95 possibilidades de valor**

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

SUBIDA DE ENCOSTA

- 95^{13} = **5.13×10^{25}** iterações
- Supondo que o código em Python consiga executar cerca de **4 milhões de iterações por segundo**
- **$5.13 \times 10^{25} / 4 \times 10^{6} = 1.2825 \times 10^{19}$ segundos**
  - **Um ano tem 31.536.000 segundos = 3.1536×10^{7}**
- **$1.2825 \times 10^{19} / 3.1536 \times 10^{7}$ anos ≅ 4.067×10^{11} anos**
- **ou aproximadamente 406 bilhões de anos**

SUBIDA DE ENCOSTA

- Agora, se existir algum tipo de *feedback* do jogo que nos diz se nossas tentativas estão mais próximas ou não do objetivo, podemos utilizar outras estratégias além da força bruta!

- Como a Subida de Encosta!

SUBIDA DE ENCOSTA

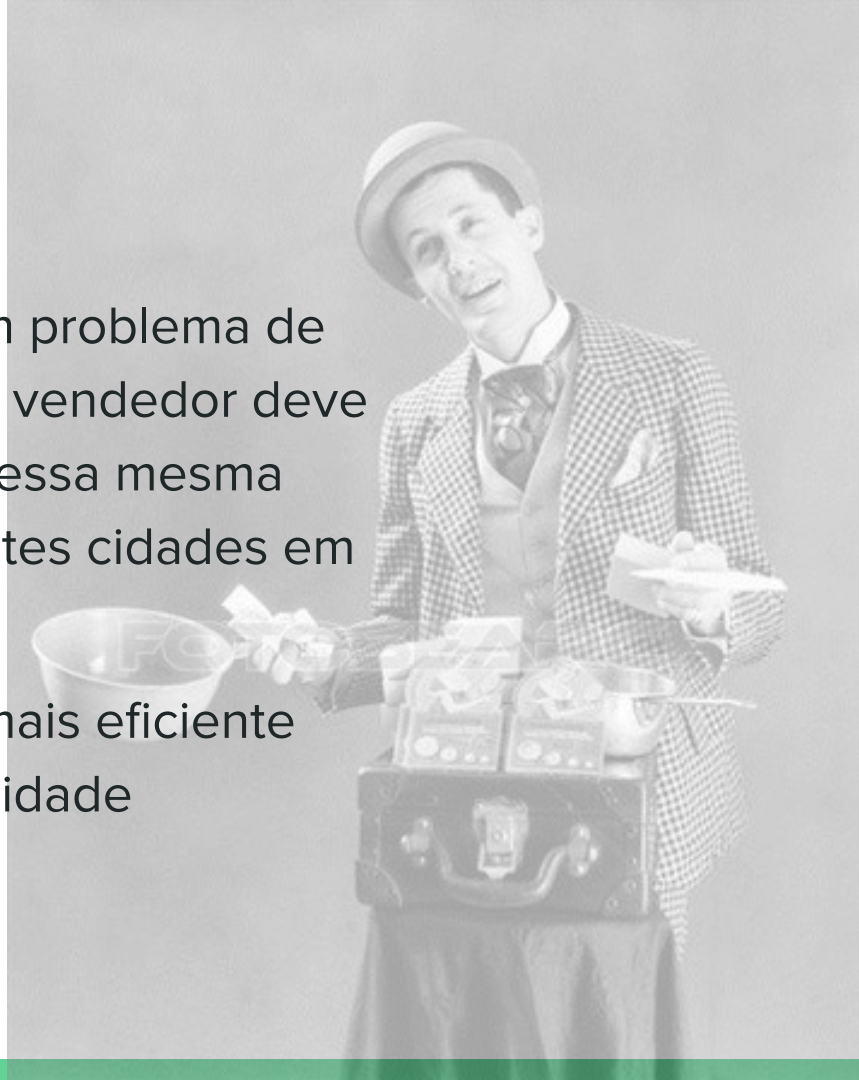
- Vamos considerar que a frase a ser descoberta é ***Hello, World!***
- Código em aula

PROBLEMA DO CAIXEIRO VIAJANTE

Problema do Caixeiro Viajante

The Travelling Salesman Problem - TSP

- O **problema do caixeiro-viajante** é um problema de *otimização combinatorial*, em que um vendedor deve começar em uma cidade e retornar a essa mesma cidade depois de viajar para ***n*** diferentes cidades em uma lista
- A questão é saber qual é o caminho mais eficiente (menor rota) para percorrer todas as cidade



Problema do Caixeiro Viajante

The Travelling Salesman Problem - TSP

- O problema se torna exponencialmente mais difícil conforme mais cidades são adicionadas à lista
- Possibilidades de rotas para ***n*** cidades ***R(n)***:
  - ***R(n) = (n-1)!***

Fonte: http://www.mat.ufrgs.br/~portosil/caixeiro.html

Problema do Caixeiro Viajante

The Travelling Salesman Problem - TSP

- Com 2 cidades:
  - ***R(2) = (2-1)! = 1 possibilidade de rota***
- Com 3 cidades:
  - ***R(3) = (3-1)! = 2 possibilidades de rotas***
- Com 4 cidades:
  - ***R(4) = (4-1)! = 6 possibilidades de rotas***

Problema do Caixeiro Viajante

The Travelling Salesman Problem - TSP

- Exemplo: Como percorrer todas as capitais do Brasil, começando de São Paulo (e voltando para São Paulo no fim)
- R(27) = (27-1)!
- **R(27) ≅ $4x10^{26}$**

Problema do Caixeiro Viajante

The Travelling Salesman Problem - TSP

- **R(27) ≅ $4x10^{26}$ rotas**
- Supondo que o computador consiga analisar 4 milhões de rotas por segundo
- $4x10^{26}$ / $4x10^{6}$ ≅

  $1x10^{20}$ segundos
- **3.17×10^{12} anos**
- **mais que 3 trilhões de anos**

SUBIDA DE ENCOSTA

- Código em aula

RECOZIMENTO SIMULADO
SIMULATED ANNEALING

SIMULATED ANNEALING

- Embora existam variantes que podem melhorar a ***subida de encosta***, todas compartilham a mesma falha: ***quando o algoritmo atinge um máximo local, ele para de funcionar***

- O **Recozimento Simulado** permite que o algoritmo não fique preso se cair em um máximo local

SIMULATED ANNEALING

- O **Recozimento Simulado** é um algoritmo **estocástico** de otimização
  - Padrão estocástico é aquele cujo estado é indeterminado, com origem em eventos aleatórios
  - Normalmente, qualquer sistema ou processo analisado com a teoria da probabilidade é estocástico
  - Por exemplo, o lançamento de uma dado resulta num processo estocástico, pois qualquer uma das 6 faces têm iguais probabilidades de ficar para cima

SIMULATED ANNEALING

- O núcleo do método de Recozimento Simulado é a analogia com a termodinâmica, especificamente com a maneira como os metais resfriam e recozem

- A essência do processo de recozimento é o resfriamento lento, que garante que um estado de baixa energia seja alcançado

- Quando um metal líquido é resfriado rapidamente *(ou temperado)*, ele acaba e um estado policristalino, com energia mais alta

SIMULATED ANNEALING

- A analogia é a seguinte:
    - Se buscarmos otimizações gananciosas, que buscam encontrar a solução o mais rápido possível *(resfriamento rápido)*, é bem provável que máximos ou mínimos locais sejam encontrados
    - Resfriamentos mais lentos *(recozimento)*, podem levar a máximos ou mínimos globais

SIMULATED ANNEALING

- No processo do Recozimento Simulado, utilizamos a seguinte metáfora:
  - No início, a temperatura é alta, sendo mais provável a tomada de decisões aleatórias
  - Conforme a temperatura diminui, torna-se menos provável a tomada de decisões aleatórias

SIMULATED ANNEALING

- Este mecanismo permite que o algoritmo mude seu estado para um vizinho que, eventualmente, pode ser pior do que o estado atual, que é como ele pode escapar dos máximos/mínimos locais

SIMULATED ANNEALING

- Por ser estocástico, o Recozimento Simulado toma decisões de usar ou não vizinhos que pioram a solução geral baseados na distribuição de probabilidade de Boltzman:
    - ***Prob(E) ~ exp(-E/kT)***
    - **E** é a energia
    - **T** é temperatura
    - **k** é uma constante que relaciona Energia e Temperatura

SIMULATED ANNEALING

- Código em aula: exemplo com o Problema do Caixeiro Viajante (TSP)

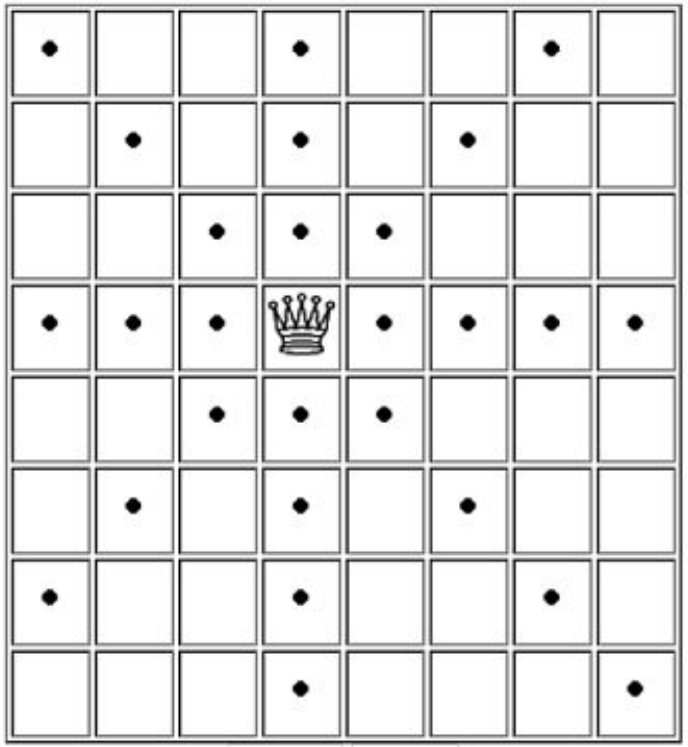
Exercício para Entrega

EXERCÍCIO - OTIMIZAÇÃO

- ***n* rainhas:**
- O problema das ***n*** rainhas consiste em posicionar ***n*** rainhas em um tabuleiro de mesa ***n*** x ***n*** sem que nenhuma delas esteja na linha de ataque entre si
- As rainhas podem se mover horizontalmente, diagonalmente e verticalmente

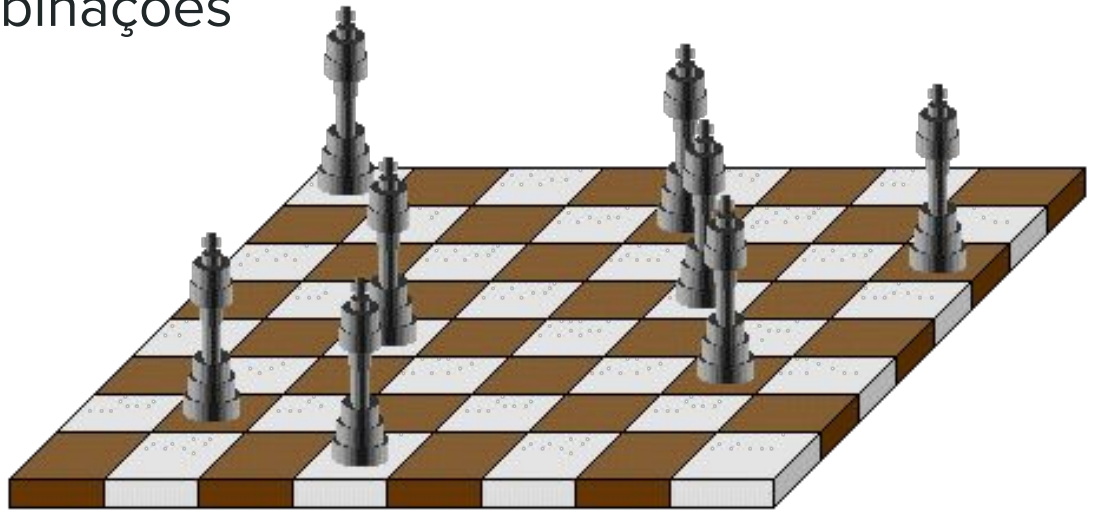
EXERCÍCIO - OTIMIZAÇÃO

- ***Exemplo: 8* rainhas**
- Possíveis movimentações da rainha:



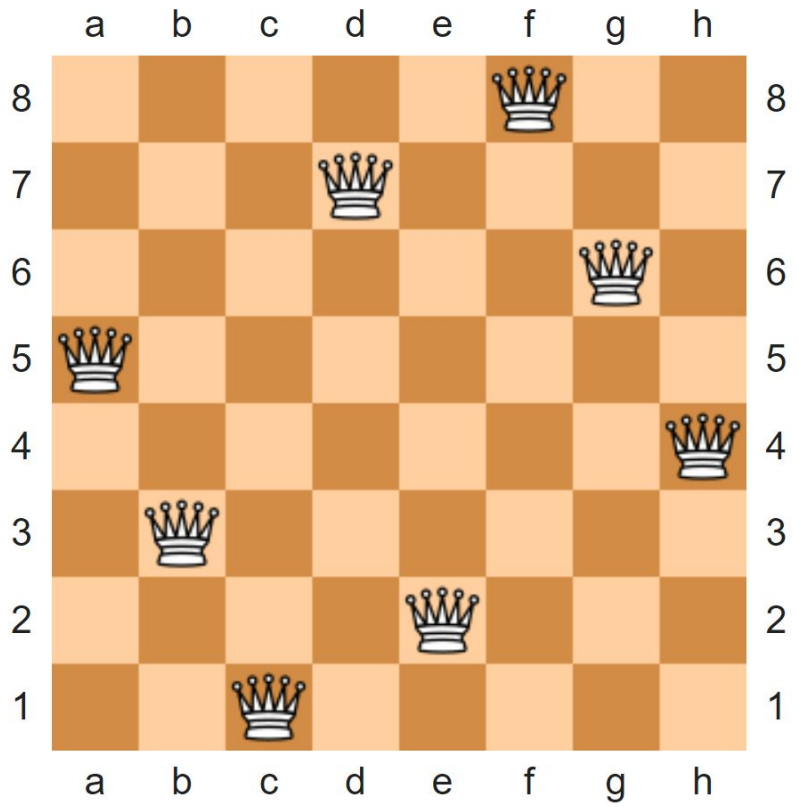
EXERCÍCIO - OTIMIZAÇÃO

- ***Exemplo: 8* rainhas**
- Encontrar a solução para o problema não é fácil, pois existe um grande número de combinações



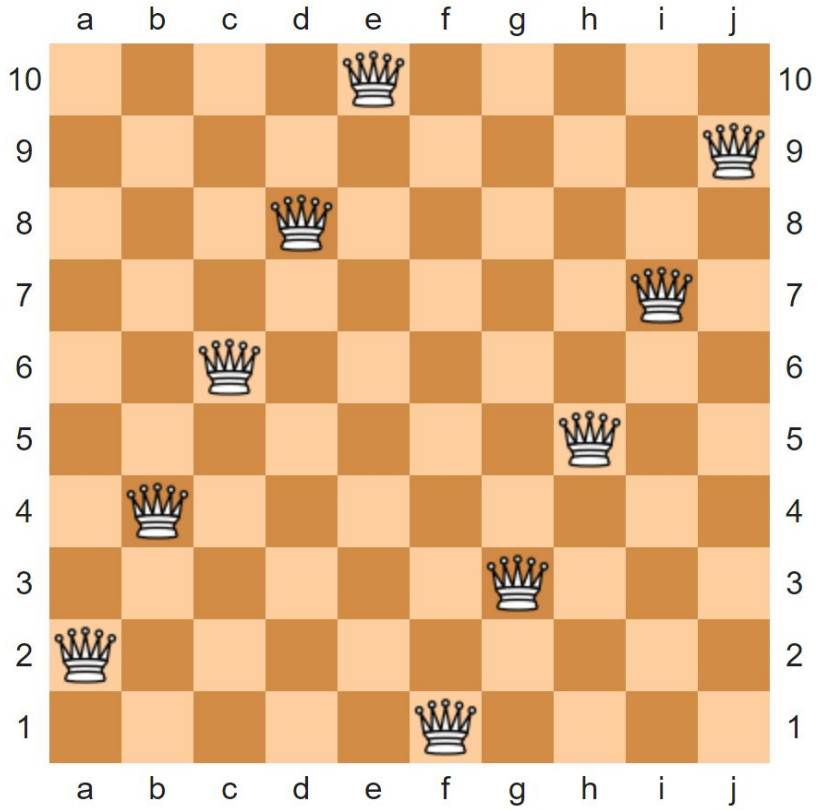
EXERCÍCIO - OTIMIZAÇÃO

- ***Exemplo: 8* rainhas**
- O problema com 8 rainhas apresenta 4.426.165.368 possíveis disposições das 8 rainhas *combinação: C(64,8) = 64!/((64-8)!x(8!))*
- Porém, somente 92 soluções
- Uma possível solução pode ser vista ao lado



ENTREGA - OTIMIZAÇÃO

- Encontrar **uma** solução do problema com ***10 rainhas (tabuleiro 10x10)***
- C(100,10) = 100!/((100-10)!x(10!)) = **1.73×10^{13}**
- Existem 724 soluções nesse caso



ENTREGA - OTIMIZAÇÃO

- Utilize as seguintes abordagens no problema:
  - Força Bruta
  - Subida de Encosta
  - Recozimento Simulado
- Compare todos os métodos com relação à quantidade de iterações e tempo para encontrar uma solução

ENTREGA - OTIMIZAÇÃO

- Entregue o código comentado em Python:
  - Arquivo ***.ipynb***