

Exemplos

- Jogo dos oito :-)
- Mundo dos blocos (ex: torre de Hanoi)
- Problema das n-rainhas
- Criptoaritmética
- Missionários e Canibais
- Resta-um
- e muitos outros...

Rainhas

- 2 formas de se resolver o problema: incremental e completo
- Possíveis estados e jogadas:
 - a) qq arranjo de 0 a 8 rainhas no tabuleiro
 - b) adicionar 1 rainha a qq posição do tabuleiro
 - a) arranjos de 0 a 8 rainhas com nenhuma em posição de ataque
 - b) colocar 1 rainha na próxima coluna vazia mais à esquerda de forma que não seja atacada por outra rainha
 - a) arranjos de 8 rainhas, 1 em cada coluna
 - b) mover qq rainha atacada para outra posição na mesma coluna

Criptoaritmética

SOLUCAO

FORTY	29786	com: F = 2	
+ TEN	850	O = 9	
+ TEN	850	R = 7 etc	

SIXTY	31486		

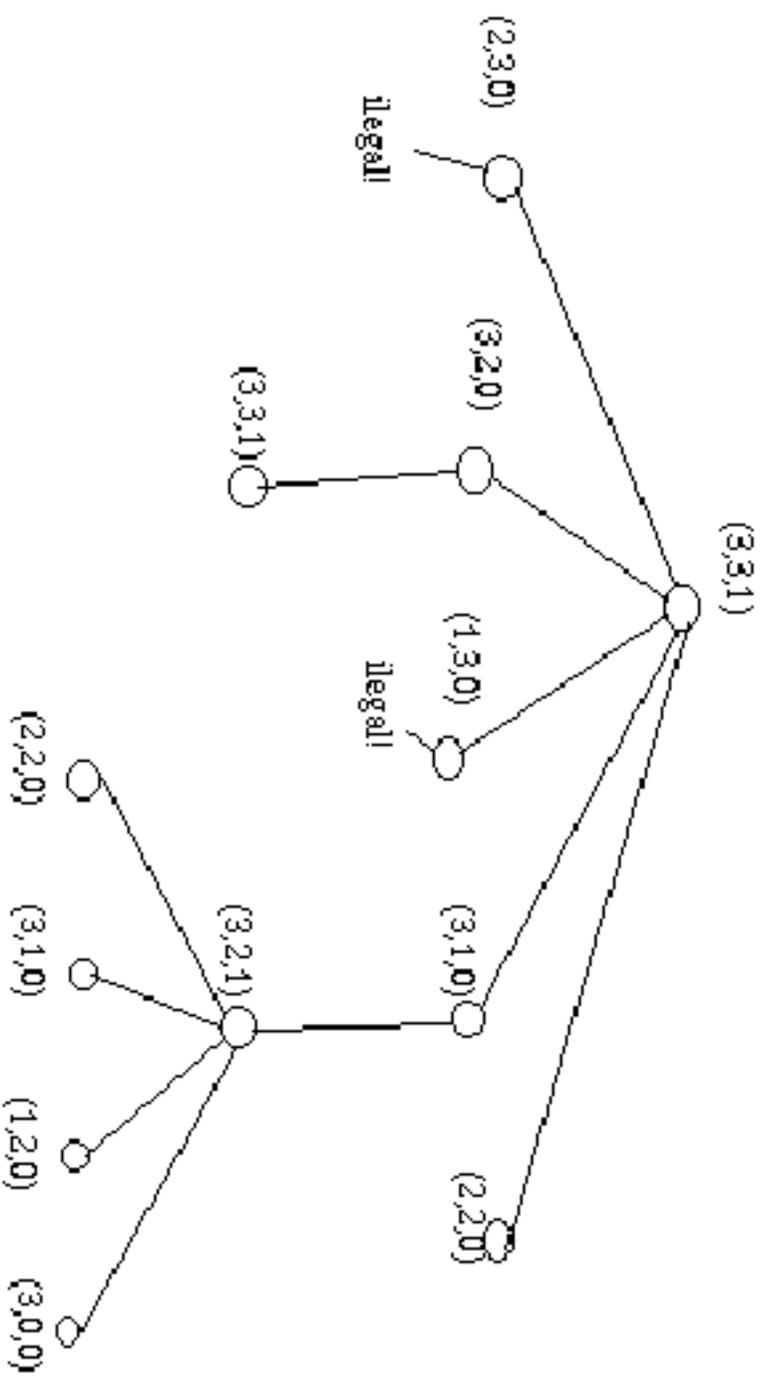
Criptaritmética

- estados: conjunto de letras que são substituídas por dígitos
- operadores: substituir todas as ocorrências de uma letra por um dígito que ainda não tenha aparecido
- possíveis regras de seleção dos dígitos: ordem alfabética para evitar repetições, seleção mais restrita respeitando as operações aritméticas
- estado final: jogo contém somente dígitos e representa uma soma correta
- custo da solução: zero

Missionários e Canibais

- 3 missionários e 3 canibais estão às margens de um rio com um bote que cabe no máximo 2 pessoas.
- Problema: encontrar uma forma de passar todos para a outra margem sem nunca deixar um grupo de missionários em uma margem com um número maior de canibais.
- estados: qq seqüência ordenada de 3 números representando o no. de missionários, canibais e botes na margem do rio.
- estado inicial: $(3,3,1)$
- estado final: $(0,0,0)$
- custo: número de travessias no rio.
- regras: tirar 1 missionário, tirar 1 canibal, tirar 2 canibais etc...

Missionários e Canibais



Sistemas de Produção

- Elementos principais de um sistema de produção em IA:
 - bancos de dados global
 - regras de produção ou restrições
 - sistema de controle

Estratégias de controle

- irrevogáveis: nunca retornam por um caminho já explorado
- tentativa: “backtracking” (métodos não informados e informados).

Algoritmos de Melhoramento Iterativo

- Utilizados para problemas para os quais a descrição já contém toda a informação para encontrar a solução (ex: n-rainhas e layout de circuitos VLSI)
- parte-se de uma config inicial conhecida e tenta-se melhorar a solução.

Algoritmos de Melhoramento Iterativo

- Exemplos:
 - Método Irreversível: Hill-Climbing (busca de máximo/mínimo local)
 - Métodos de tentativa: (busca de máximo/mínimo global) podem temporariamente tornar estados piores.
 - * random restart-hill-climbing
 - * simulated annealing
 - * busca tabu
 - * busca paralela

Hill Climbing (ou gradiente descendente)

- tenta fazer modificações que melhorem o estado corrente
- 2 desvantagens:
 - máximos locais
 - platôs: percorre estados aleatoriamente porque a função de avaliação não muda muito
- Exemplo: Jogo dos oito :-)

Hill Climbing: exemplo

2 8 3	2 8 3	2 3	2 3	1 2 3	1 2 3
1 6 4	1 4	1 8 4	1 8 4	8 4	8 4
7 5	7 6 5	7 6 5	7 6 5	7 6 5	7 6 5

$f=-4$ $f=-3$ $f=-3$ $f=-2$ $f=-1$ $f=0$

- No caso de não se poder aplicar a regra, o processo termina.
- Solução = máximo local, hill-climbing não encontra máximo global.

Hill Climbing: exemplo

- não funciona para:

1 2 5		1 2 3
7 4	=>	7 4
8 6 3		8 6 5

$f=-2$

- Qq movimento diminui o valor da função de avaliação.

Random Restart Hill Climbing

- executa uma série de buscas hill-climbing a partir de estados iniciais aleatórios
- cada um roda até terminar ou até não ter nenhum progresso
- salva o melhor resultado obtido até então
- pode ter no. finito de iterações ou continuar até não conseguir melhorar o melhor valor encontrado
- se superfície de busca contém muitos máximos locais, busca exponencial
- geralmente, solução boa pode ser encontrada em um número pequeno de iterações.

Simulated Annealing

- Invés de começar novamente aleatoriamente quando passa num máximo local, permite que a busca escape do máximo local “descendo a montanha”.

Simulated Annealing

```
function SA(problem, schedule) return a solution state
current <- MAKE_NODE(INITIAL_STATE[problem])
for t <- 1 to infinito do
  T <- schedule(t)
  if T = 0 then return current
  next <- successor de current seleccionado
    aleatoriamente
  deltaE <- value[next] - value[current]
  if deltaE > 0 then current <- next
  else current <- next with prob  $e^{-\text{deltaE}/T}$ 
endif
endfor
```

Simulated Annealing

```
P = e-(deltaE/T)  
n = sorteio de um no. de 0 a 1  
if n < P then current <- next
```

- ou seja: quanto maior a prob mais chance de aceitar mover para passos de custo pior.

Outros Algoritmos

- Algoritmos Genéticos
 - Operações: “crossover”, mutação e reprodução
 - começa de uma população inicial
 - aplica as operações
 - calcula uma função de “fitness” para cada indivíduo da população
 - pode eliminar indivíduos menos “fit”

Outros Algoritmos

- Satisfação de Restrições
 - tipo especial de problema que satisfaz propriedades estruturais além dos requisitos básicos para problemas gerais de busca
 - estados: conjunto de variáveis
 - domínio: conjunto possível de valores que uma variável pode assumir (discreto/contínuo, finito/infinito)
 - estado inicial: todas as variáveis com valores possíveis iniciais
 - estado final: valores finais para as variáveis que respeitem as restrições do problema
 - profundidade máxima da árvore: número de variáveis

Satisfação de Restrições: Exemplo

- n-rainhas
 - variáveis: posições no tabuleiro
 - restrições: nenhuma rainha pode atacar outra na mesma linha, diagonal ou coluna
 - valores iniciais das variáveis: V_1 in $1..n$, V_2 in $1..n$ etc, com n largura do tabuleiro

Satisfação de Restrições: Possíveis algoritmos

- aloca uma nova rainha para uma nova coluna a cada nível (solução incremental)
- complexidade: $n^n \approx \pi D_i = D_1 \times D_2 \times \dots \times D_n$
- fator de ramificação: n
- fator máximo de ramificação:
$$n \times n = \sum D_i = D_1 + D_2 + \dots + D_n$$
- se todas as variáveis fossem instanciadas com todos os valores possíveis no primeiro nível da árvore

Satisfação de Restrições: Possíveis algoritmos

$n = 8$

(0,0,0,0,0,0,0,0)

N	(1,0,0,0,0,0,0,0)	(0,1,0,0,0,0,0,0)	(0,0,1,0,0,0,0,0)
i	(2,0,0,0,0,0,0,0)	(0,2,0,0,0,0,0,0)	(0,0,2,0,0,0,0,0)
v			
1	(8,0,0,0,0,0,0,0)	(0,8,0,0,0,0,0,0)	(0,0,8,0,0,0,0,0)
N	(1,1,0,0,0,0,0,0)	(0,1,1,0,0,0,0,0)	(1,0,1,0,0,0,0,0)
i	(2,2,0,0,0,0,0,0)	(0,2,2,0,0,0,0,0)	(2,0,2,0,0,0,0,0)
v			
2	(8,8,0,0,0,0,0,0)	(0,8,8,0,0,0,0,0)	(8,0,8,0,0,0,0,0)

Satisfação de Restrições: Possíveis algoritmos

- utilizando BP a sub-árvore que contém
 - $(0, 0, 0, 0, 0, 0, 0, 0) \rightarrow (1, 0, 0, 0, 0, 0, 0, 0) \rightarrow$
 - $(1, 1, 0, 0, 0, 0, 0, 0) \rightarrow$
 - $(1, 1, 1, 0, 0, 0, 0, 0) \rightarrow (1, 1, 1, 1, 0, 0, 0, 0) \rightarrow \dots$
- será explorada mesmo sabendo que $(1, 1, 1, 1, 1, 1, 1, 1)$ não é solução

Satisfação de Restrições: Possíveis algoritmos

- solução: colocar o teste de restrição a cada rainha colocada no tabuleiro
- tb não é a melhor solução!
- Suponha que já conseguimos alocar 6 rainhas, mas esta alocação ataca a oitava rainha.
- BP testa todas as possibilidades de colocação da sétima rainha!

Satisfação de Restrições: Possíveis algoritmos

- solução: algoritmos *Forward Checking* e *Lookahead* ou simplesmente “consistência de arcos”
- *Forward Checking*: retira dos domínios de outras variáveis todos os valores impossíveis que violam as restrições
- *Lookahead*: além de executar *forward checking*, verifica se o domínio modificado de cada variável conflita com os outros.

Forward Checking: Exemplo

$n = 8$

(1) $V1 = 1 \implies V2 = \{3, 4, 5, 6, 7, 8\}$
 $V3 = \{2, 4, 5, 6, 7, 8\}$
 $V4 = \{2, 3, 5, 6, 7, 8\}$
 $V5 = \{2, 3, 4, 6, 7, 8\}$
 $V6 = \{2, 3, 4, 5, 7, 8\}$
 $V7 = \{2, 3, 4, 5, 6, 8\}$
 $V8 = \{2, 3, 4, 5, 6, 7\}$

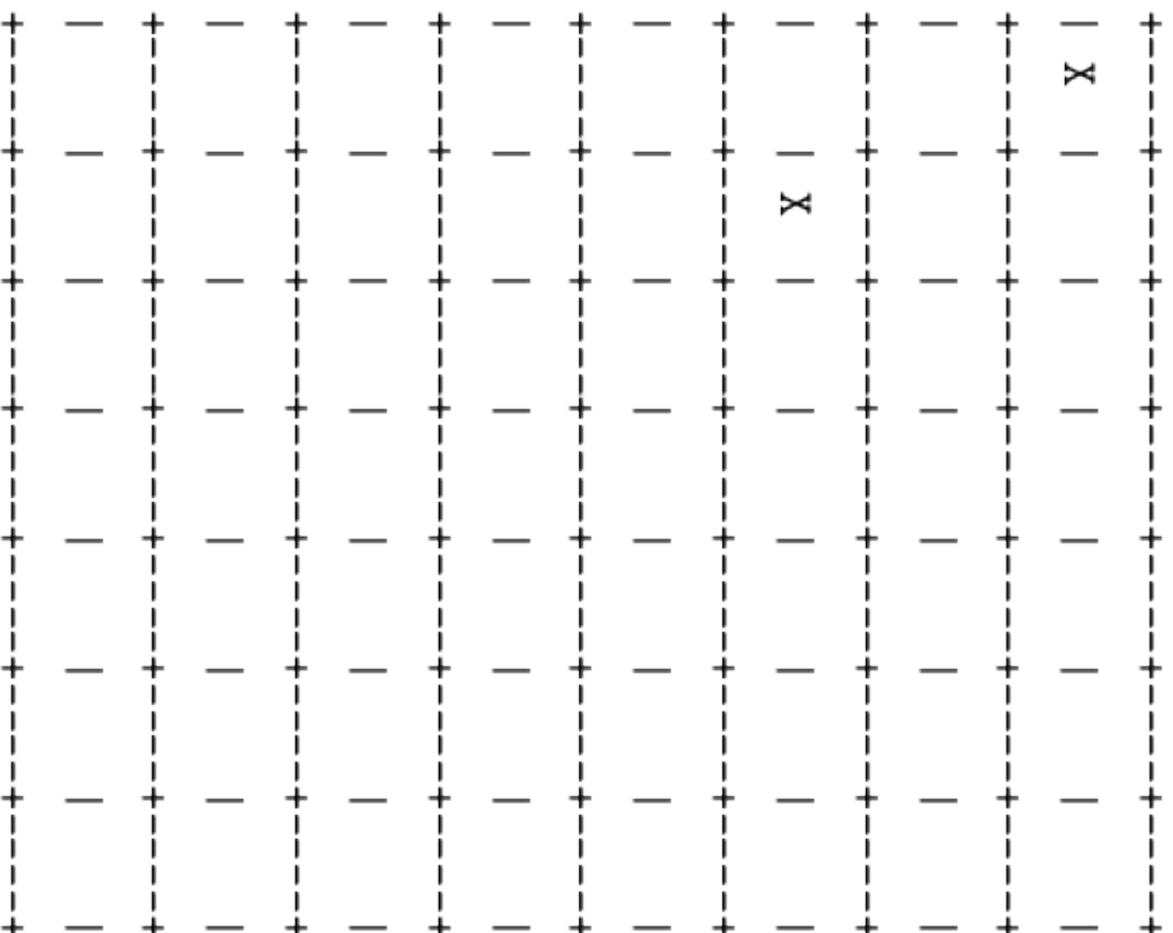
(2) $V2 = 3 \implies V3 = \{5, 6, 7, 8\}$
 $V4 = \{2, 6, 7, 8\}$
 $V5 = \{2, 4, 7, 8\}$
 $V6 = \{2, 4, 5, 8\}$
 $V7 = \{2, 4, 5, 6\}$
 $V8 = \{2, 4, 5, 6, 7\}$

n-rainhas

n=8

apos V1=1

V2=3



Satisfação de Restrições: Possíveis algoritmos

- para domínios infinitos: programação linear, simplex, revised simplex, convex hull, eliminação de Gauss.
- para domínios finitos: forward checking, lookahead, consistência de arcos em geral.
- Para domínios finitos, dois problemas:
 - escolha da *variável*:
 - * *most-constrained*: menor domínio
 - * *most constraining*: restringe ao máximo os domínios das outras variáveis
 - escolha do *valor* da variável:
 - * princípio *first-fail*.
 - * *least constraining*: valor que afeta menos o conjunto de valores das outras variáveis

Coloracao de Mapas : 3 cores +-----+

azul | B |

verde +-----+ |

vermelho | A | |

| | | |

+-----+ +-----+ |

| C +-----+ |

| | | |

+-----+ +-----+ |

| | +-----+ |

| | E | |

| +-----+ +-----+ |

| | | |

| | | |

| | | |

| F | D |

+-----+ +-----+ |

Satisfação de Restrições

- variável most-constrained: permite resolver n-rainhas com n igual a 100.
- forward checking puro: só consegue resolver até 30.
- valor least-constraining: permite resolver n-rainhas com n igual a 1000.