



Inteligência Artificial

Prof. Tiago A. E. Ferreira

Aula 11 – Problemas com Satisfação
de Vínculos



Constraint Satisfaction Problems (CSP)

- Conceitos básicos
- Busca cega simples e refinada
- Busca heurística
- CSP iterativo

Constraint Satisfaction Problems (CSP)

- Um Problema de Satisfação de Restrições
 - tipo de problema que impõe propriedades estruturais adicionais à solução a ser encontrada
 - há uma demanda mais refinada do que na busca clássica
 - ex. ir de Recife à Cajazeiras com no máximo 3 tanques de gasolina e 7 horas de viagem
- Um CSP consistem em:
 - um conjunto de variáveis que podem assumir valores dentro de um dado domínio
 - um conjunto de restrições que especificam propriedades da solução - valores que essas variáveis *podem* assumir.



CSP: Formulação

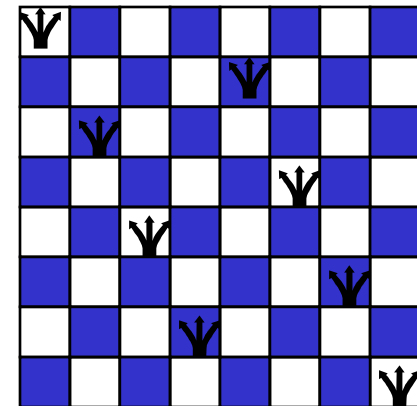
- Estados:
 - definidos pelos valores possíveis das variáveis
- Estado inicial:
 - nenhuma variável instanciada ainda
- Operadores:
 - atribuem valores (instanciação) às variáveis (uma variável por vez)
- Teste de término:
 - verificar se todas as variáveis estão instanciadas obedecendo as restrições do problema
- Solução:
 - conjunto dos valores das variáveis instanciadas
- Custo de caminho:
 - número de passos de atribuição

CSP: características das restrições

- O conjunto de valores que a variável i pode assumir é chamado domínio D_i
 - O domínio pode ser **discreto** (fabricantes de uma peça do carro) ou **contínuo** (peso das peças do carro)
- Quanto à aridade, as restrições podem ser
 - unárias (sobre uma única variável)
 - binárias (sobre duas variáveis) - ex. 8-rainhas
 - n-árias - ex. palavras cruzadas
 - a restrição unária é um sub-conjunto do domínio, enquanto que a n-ária é um produto cartesiano dos domínios
- Quanto à natureza, as restrições podem ser
 - **absolutas** (não podem ser violadas)
 - **preferenciais** (devem ser satisfeitas quando possível)

Exemplo

- Problema das 8-rainhas
 - variáveis: localização das rainhas
 - valores: possíveis posições do tabuleiro
 - restrição binária: duas rainhas não podem estar na mesma coluna, linha ou diagonal
 - solução: valores para os quais a restrição é satisfeita





Busca cega para CSP

■ Funcionamento

- estado inicial: variáveis sem atribuição
- aplica operador: instancia **uma** variável
- teste de parada: todas variáveis instanciadas sem violações

■ Análise

- pode ser implementada com **busca em profundidade limitada** (l = número de variáveis)
- é completa
- fator de expansão: $\sum_i |D_i|$ (Cardinalidade do conj. de valores pra a variável i)
- o teste de parada é decomposto em um conjunto de restrições sobre as variáveis

Exemplo: coloração de mapas

Simulação passo a passo...

A= green

B = green

C= green

D=green

E=green

F=green (falha c/ C, E, D)

F=red

E (falha c/ C,A,B)

E=red (falha c/ F)

E=blue

C (falha c/ A)

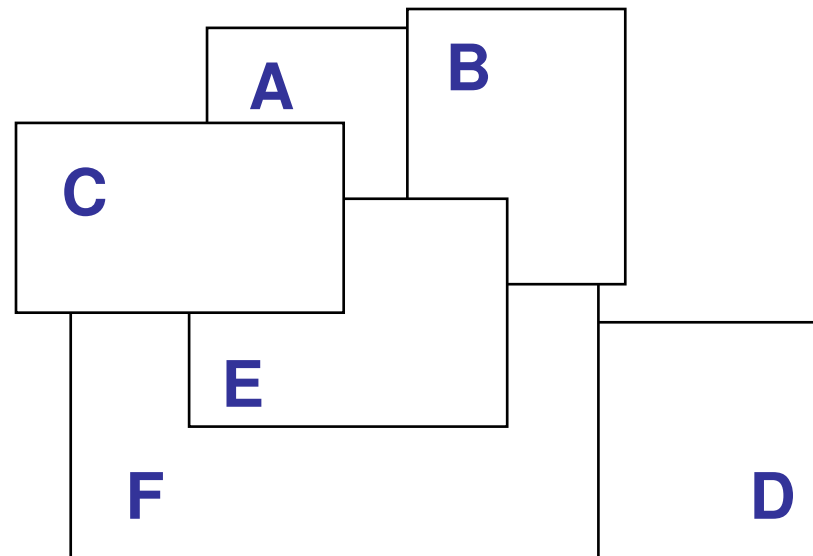
...

Muito dispendioso

variáveis: A,B,C,D,E,F

domínio: {green,red,blue}

restrições: $A \neq B$; $A \neq C$; $A \neq E$;
 $B \neq E$; $B \neq F$; $C \neq E$;
 $C \neq F$; $D \neq F$; $E \neq F$





Backtracking na busca cega

- Problema da busca em profundidade
 - perda de tempo pois continua mesmo que uma restrição já tenha sido violada (não se pode mais redimir o erro)
- Solução: Backtracking
 - depois de realizar uma atribuição, verifica se restrições não são violadas
 - caso haja violação \Rightarrow backtrack

Exemplo: coloração de mapas

Simulação passo a passo...

A= green

B = green (falha c/ A)

B=red

C=green (falha c/ A)

C= red

D=green

E= green (falha c/ A)

E= red (falha c/ B e C)

E= blue

F=green (falha c/ D)

F=red (falha c/ C)

F = blue (falha c/ E)

F backtracking

E backtracking

D=red

E=green (falha c/ A)

E= red (falha c/ B)

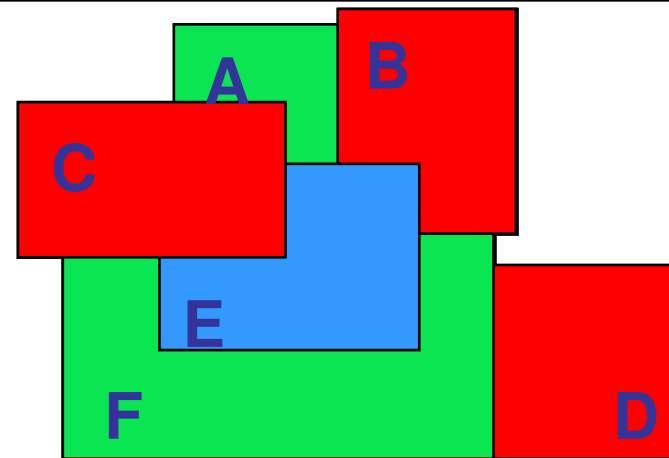
E= blue

F=green

variáveis: A,B,C,D,E,F

domínio: $D_a=D_b\dots=D_f=\{\text{green,red,blue}\}$

restrições: $A \neq B; A \neq C; A \neq E; B \neq E;$
 $B \neq F; C \neq E; C \neq F; D \neq F;$
 $E \neq F$



Exemplo: coloração de mapas

Mas poderia ser mais complicado começando por red...

A=red

B=green

C=blue

D=red

E= ?? Backtracking

D=green

E=?? Backtracking

D=blue

E=?? Backtracking

D= ?? Backtracking

C = green

D = green

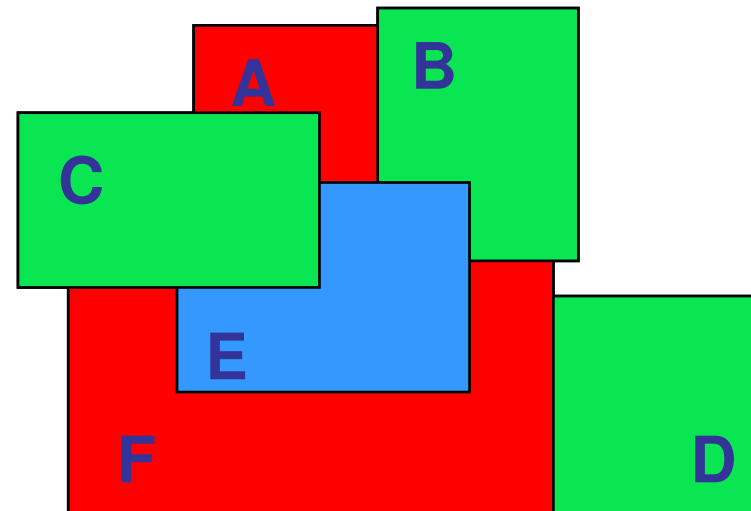
E = blue

F=red

variáveis: A,B,C,D,E,F

domínio: $D_a=D_b\dots=D_f=\{\text{green,red,blue}\}$

restrições: $A \neq B; A \neq C; A \neq E; B \neq E;$
 $B \neq F; C \neq E; C \neq F; D \neq F;$
 $E \neq F$





Backtracking não basta...

- Problema do backtracking:
 - não adianta mexer na 7a. rainha para poder posicionar a última.
 - O problema é mais em cima... O *backtrack* tem que ser de mais de um passo
- Soluções
 - Verificação de arco-consistência (*forward checking*)
 - Propagação de restrições



Refinamentos

- Verificação prévia (*forward checking*)
 - **idéia**: olhar para frente para detectar situações insolúveis
 - ex. no restaurante self-service
- Algoritmo:
 - Após cada atribuição, elimina do domínio das variáveis não instanciadas os valores incompatíveis com as atribuições feitas até agora.
 - Se um domínio torna-se vazio, *backtrack* imediatamente.
- É bem mais eficiente!



Propagação de restrições

- *Forward checking* é um caso particular de verificação de arco-consistência
 - um estado é arco-consistente se o valor de cada variável é consistente com as restrições sobre esta variável
 - arco-consistência é obtida por sucessivas eliminações de valores inconsistentes
- Propagação de restrições (*constraint propagation*)
 - uma consequência da verificação de arco-consistência
 - quando um valor é eliminado, outros podem se tornar inconsistentes e tendo que ser descartados.
 - é como uma onda que se propaga: as escolhas ficam cada vez mais restritas

Propagação de restrições

Exemplo: coloração de mapas

Passo a passo...

Variáveis = A, B, C, D, E, F
Domínios = {red, green, blue}

A=red

=> B, C, E = {green, blue}

(variáveis c/ restrições c/ A)

=> D, F = {red, green, blue}

B=green

=> E = {blue}, F = {red, blue} (variáveis c/ restrições c/ B)

=> C = {green, blue}, D = {red, green, blue}

C = green

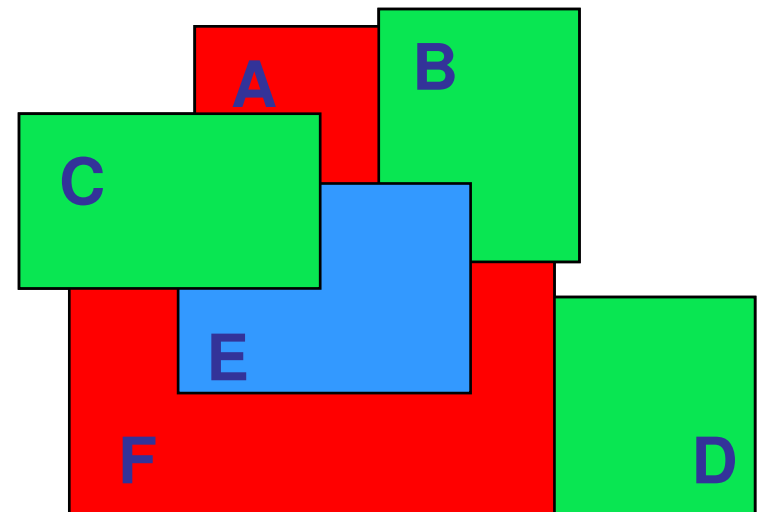
=> E = {blue}, F = {red, blue} (restrições c/ C)

=> D = {red, green, blue}

D=red, E=blue, F=??

Backtracking!!

D=green, E=blue, F=red





Heurísticas para CSP

- Tenta reduzir o fator de expansão do espaço de estados
- Onde pode entrar uma heurística?
 - Ordenando a escolha da **variável** a instanciar
 - Ordenando a escolha do **valor** a ser associado a uma variável
- Existem 3 heurísticas para isto...
 - variável mais restritiva: variável envolvida no maior número de restrições é preferida
 - Variável mais restringida: variável que pode assumir menos valores é preferida
 - valor menos restritivo: valor que deixa mais liberdade para futuras escolhas

Variável mais restritiva

(variável envolvida no maior número de restrições)

Candidatas¹: E, F, ...resto

E = green

Candidatas: F, ...resto

F = red

Candidatos: A, B, C, D

A = red

Candidatos: B, C, D

B = blue

Candidatos: C, D

C = blue

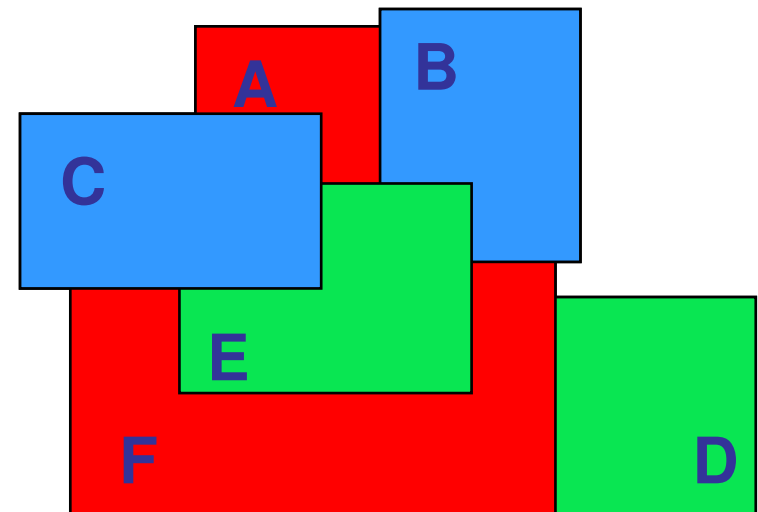
D = green

SEM BACKTRACK!!

variáveis: A,B,C,D,E,F

domínio: Da=Db...=Df={green,red,blue}

restrições: A ≠ B; A ≠ C; A ≠ E; B ≠ E; B ≠ F; C ≠ E; C ≠ F; D ≠ F; E ≠ F



¹ em ordem de prioridade

Coloração de mapas: variável mais restringida

(variável que pode assumir menos valores)

Candidatas: todas

A = green

Candidatas: B, C, E, ...

B = red

Candidatos: E, F, ...

E=blue

Candidatos: C, F, D

C=red

Candidatos: F, D

F=green

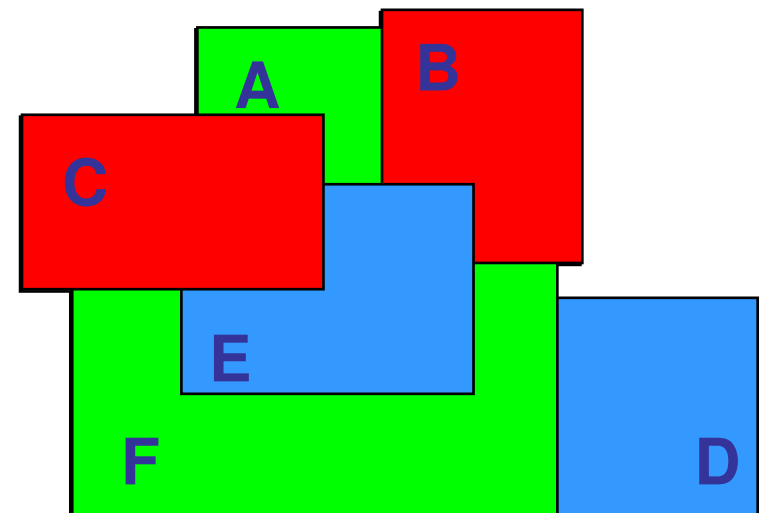
D = blue ou red

SEM BACKTRACK!!

variáveis: A,B,C,D,E,F

domínio: $D_a=D_b\dots=D_f=\{\text{green,red,blue}\}$

restrições: $A \neq B; A \neq C; A \neq E; B \neq E; B \neq F; C \neq E; C \neq F; D \neq F; E \neq F$



Coloração de mapas: valor menos restritivo

(valor que deixa mais liberdade)

variáveis: A,B,C,D,E,F

domínio: $D_a = D_b = \dots = D_f = \{\text{green, red, blue}\}$

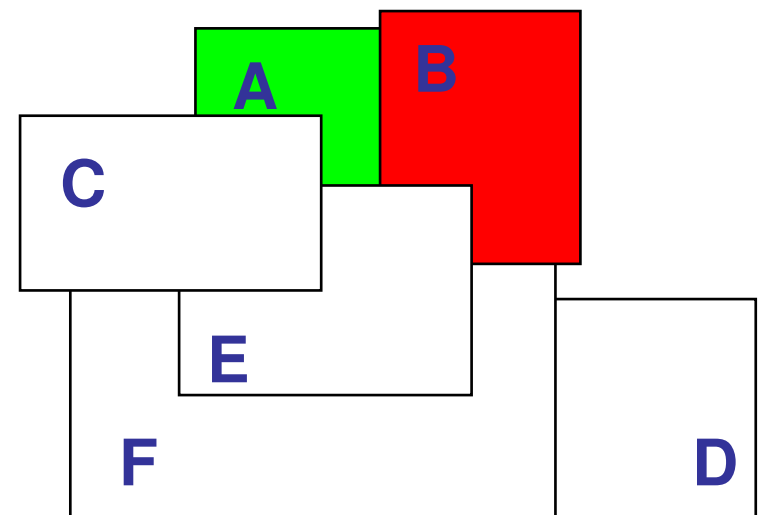
restrições: $A \neq B$; $A \neq C$; $A \neq E$; $B \neq E$; $B \neq F$; $C \neq E$; $C \neq F$; $D \neq F$; $E \neq F$

Começando com

A = green

B = red

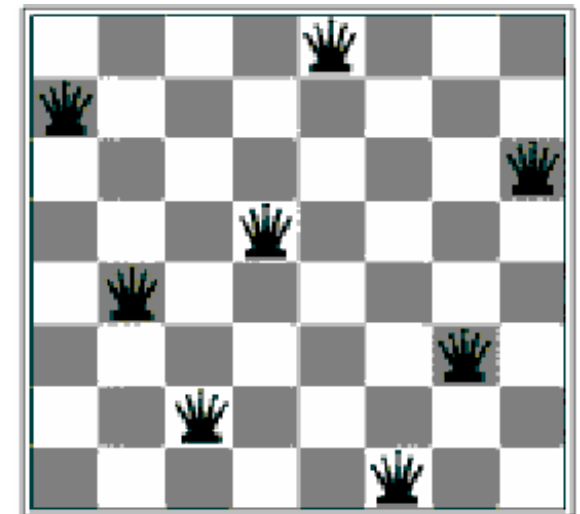
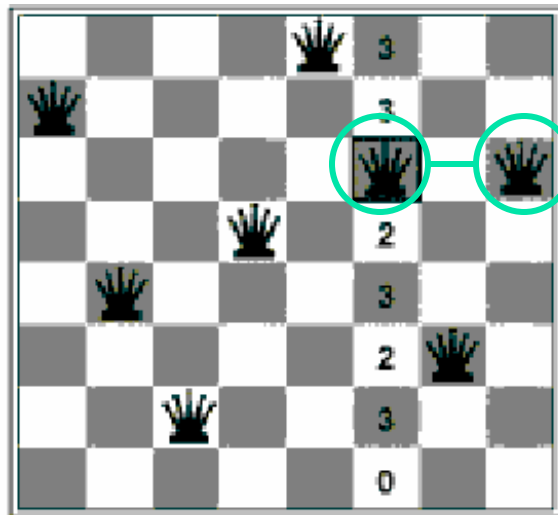
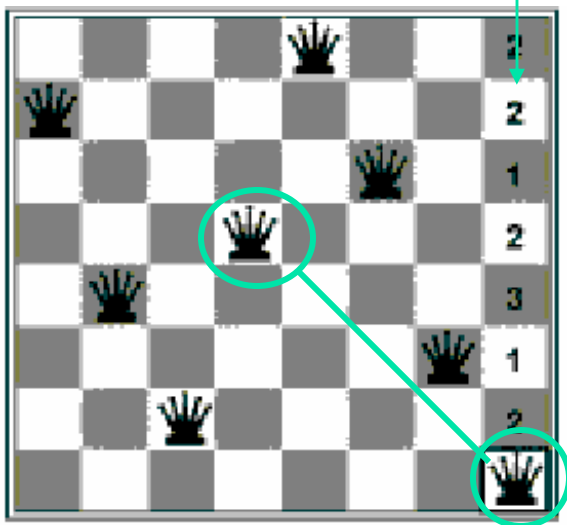
C=??? red é melhor do que **blue**



CSP iterativo

- CSP pode ser resolvido iterativamente
 - 1) instancia aleatoriamente todas variáveis
 - 2) aplica operadores para trocar os valores e então diminuir número de restrições não satisfeitas (min-conflicts).
- Heurística de reparos
 - repara inconsistências
- Min-conflict resolve 8 rainhas em menos de 50 passos!!!

Número de ataques





CSP

- Grande importância prática, sobretudo em tarefas de
 - criação (design)
 - agendamento (scheduling) ou escalonamento
 - onde várias soluções existem e é mais fácil dizer o que não se quer...
- Estado atual
 - Grandes aplicações industriais \$\$\$\$
 - Número crescente de artigos nas principais conferências
- Observação:
 - a sigla CSP também é usada para falar de Constraint Satisfaction Programming, que é um paradigma de programação