SCC-5774 - Capítulo 3
Satisfação de Restrições

João Luís Garcia Rosa[1]

[1]Departamento de Ciências de Computação
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - São Carlos
http://www.icmc.usp.br/~joaoluis

2020

Sumário

1. CSP
   - Introdução
   - Exemplos
   - Grafo de restrição
2. Variações de CSPs
   - Variáveis
   - Propagação de restrições
   - *Backtracking*
3. Estrutura do problema
   - Introdução
   - Conflitos mínimos
   - Resumo

Sumário

1. CSP
   - Introdução
   - Exemplos
   - Grafo de restrição
2. Variações de CSPs
   - Variáveis
   - Propagação de restrições
   - *Backtracking*
3. Estrutura do problema
   - Introdução
   - Conflitos mínimos
   - Resumo

CSP

- Problema de busca padrão: o estado é uma "caixa preta" $\rightarrow$ qualquer estrutura de dados antiga que suporta teste de meta, avaliação, sucessor,
- **CSP (constraint satisfaction problem)**: estado é definido pelas variáveis X_i com valores do domínio D_i,
- Teste de meta é um conjunto de restrições especificando combinações permitidas de valores para subconjuntos de variáveis,
- Exemplo simples de uma **linguagem de representação formal**,
- Permite algoritmos úteis de **propósito geral** com mais poder que os algoritmos de busca padrão.

Problemas de satisfação de restrições

- Muitos problemas em IA podem ser vistos como **problemas de satisfação de restrições** nos quais o objetivo é descobrir algum estado do problema que satisfaça um determinado conjunto de restrições.
- Exemplos desse tipo de problema incluem enigmas criptaritméticos e muitos problemas de rotulagem perceptual do mundo real.
- As tarefas de projeto também podem ser vistas como problemas de satisfação de restrições nos quais um design deve ser criado dentro de limites fixos de tempo, custo e materiais.
- Ao visualizar um problema como sendo de satisfação de restrições, é geralmente possível reduzir substancialmente a quantidade de busca necessária em comparação com um método que tenta formar soluções parciais diretamente, escolhendo valores específicos para os componentes da solução final.

Problema criptaritmético

- Por exemplo, um procedimento de busca simples para resolver um problema criptaritmético pode operar em um espaço de estados de soluções parciais, no qual as letras recebem números específicos como seus valores.
- Um esquema de controle em profundidade pode seguir um caminho de atribuições até que uma solução ou uma inconsistência seja descoberta.
- Em contraste com isso, uma abordagem de satisfação de restrições para resolver esse problema evita fazer adivinhações sobre determinadas atribuições de números para letras até que seja necessário.
- Em vez disso, o conjunto inicial de restrições, que diz que cada número pode corresponder a apenas uma letra e que as somas dos dígitos devem ser como são dadas no problema, é primeiro ampliado para incluir restrições que podem ser deduzidas das regras de aritmética.

Problema criptaritmético

- Então, embora a adivinhação ainda possa ser necessária, o número de palpites admissíveis é reduzido e, portanto, o grau de busca é reduzido.
- Satisfação de restrições é um procedimento de busca que opera em um espaço de conjuntos de restrições.
- O estado inicial contém as restrições que são originalmente fornecidas na descrição do problema.
- Um estado meta é qualquer estado que tenha sido restringido "suficientemente", onde "suficientemente" deve ser definido para cada problema.
- Por exemplo, para a criptaritmética, suficientemente significa que cada letra foi atribuída a um valor numérico exclusivo.

Propagação de restrições

- Satisfação de restrições é um processo de duas etapas.
- Primeiro, as restrições são descobertas e propagadas o máximo possível em todo o sistema.
- Então, se ainda não houver uma solução, a busca será iniciada.
- Um palpite sobre algo é feito e adicionado como uma nova restrição.
- A **propagação de restrições** pode ocorrer com essa nova restrição e assim por diante.

Propagação de restrições

- O primeiro passo, a propagação, surge do fato de que geralmente existem dependências entre as restrições.
- Essas dependências ocorrem porque muitas restrições envolvem mais de um objeto e muitos objetos participam de mais de uma restrição.
- Então, por exemplo, vamos supor que começamos com uma restrição, $N = E + 1$.
- Então, se adicionarmos a restrição $N = 3$, poderíamos propagá-la para obter uma restrição mais forte em E, ou seja, $E = 2$.
- A propagação de restrições também surge da presença de regras de inferência que permitem inferir restrições adicionais às dadas.

Propagação de restrições

- A propagação de restrições termina por um de dois motivos.
- Primeiro, uma contradição pode ser detectada.
- Se isso acontecer, então não há solução consistente com todas as restrições conhecidas.
- Se a contradição envolve apenas as restrições que foram dadas como parte da especificação do problema (ao contrário daquelas que foram adivinhadas durante a resolução), então nenhuma solução existe.
- O segundo motivo possível para a terminação é que a propagação esgotou-se e não há mais mudanças que possam ser feitas com base no conhecimento atual.
- Se isso acontecer e uma solução ainda não tiver sido adequadamente especificada, a **busca** será necessária para que o processo se mova novamente.

Busca heurística

- Neste ponto, o segundo passo começa.
- Algumas hipóteses sobre uma maneira de fortalecer as restrições devem ser feitas.
- No caso do problema criptaritmético, por exemplo, isso geralmente significa adivinhar um determinado valor para alguma letra.
- Depois disso, a propagação de restrições pode começar novamente a partir desse novo estado.
- Se uma solução for encontrada, ela poderá ser relatada.
- Se mais adivinhações forem necessárias, elas podem ser feitas.
- Se uma contradição é detectada, o *backtracking* pode ser usado para tentar um palpite diferente e continuar com ele.
- Podemos afirmar este procedimento mais precisamente pelo **Algoritmo Satisfação de Restrições**.

Algoritmo Satisfação de Restrições [1]

1. Propagar restrições disponíveis. Para fazer isso, primeiro faça ABERTOS como o conjunto de todos os objetos que devem ter valores atribuídos a eles em uma solução completa. Então faça até que uma inconsistência seja detectada ou até que ABERTOS esteja vazia:
   a. Selecione um objeto OB de ABERTOS. Fortaleça, tanto quanto possível, o conjunto de restrições que se aplicam a OB.
   b. Se este conjunto for diferente do conjunto que foi atribuído a última vez que OB foi examinado ou se esta for a primeira vez que OB foi examinado, então adicione a ABERTOS todos os objetos que compartilham quaisquer restrições com OB.
   c. Remova OB de ABERTOS.
2. Se a união das restrições descobertas acima definir uma solução, saia e relate a solução.
3. Se a união das restrições descobertas acima definir uma contradição, retorne falha.

Algoritmo Satisfação de Restrições - contd.

4. Se nenhuma das situações acima ocorrer, então é necessário adivinhar algo para prosseguir. Para fazer isso, faça um loop até que uma solução seja encontrada ou todas as soluções possíveis tenham sido eliminadas:
   a. Selecione um objeto cujo valor ainda não esteja determinado e selecione uma maneira de fortalecer as restrições nesse objeto.
   b. Recursivamente, invoque a satisfação de restrições com o conjunto atual de restrições aumentado pela restrição de fortalecimento recém-selecionada.

Regras

- Este algoritmo foi declarado o mais geral possível.
- Para aplicá-lo em um determinado domínio de problema requer o uso de dois tipos de regras: regras que definem a maneira pela qual as restrições podem ser propagadas com validade e regras que sugerem palpites quando os palpites são necessários.
- Vale a pena notar, no entanto, que em alguns domínios problemáticos a adivinhação pode não ser necessária.
- Em geral, quanto mais poderosas forem as regras para propagação de restrições, menor será a necessidade de adivinhação.

Sumário

1. CSP
   - Introdução
   - Exemplos
   - Grafo de restrição
2. Variações de CSPs
   - Variáveis
   - Propagação de restrições
   - *Backtracking*
3. Estrutura do problema
   - Introdução
   - Conflitos mínimos
   - Resumo

Problema criptaritmético

- Para ver como esse algoritmo funciona, considere o problema criptaritmético mostrado abaixo.
- O estado meta é um estado do problema no qual todas as letras receberam um dígito de forma que todas as restrições iniciais sejam satisfeitas.

Problema:

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

Estado inicial:
Não há duas letras com o mesmo valor.
As somas dos dígitos devem ser conforme mostrado no problema.

Algoritmo Satisfação de Restrições

- O processo de solução prossegue em ciclos.
- Em cada ciclo, duas coisas significativas são feitas (correspondentes aos passos 1 e 4 deste algoritmo):
  1. As restrições são propagadas usando regras que correspondem às propriedades da aritmética.
  2. Um valor é adivinhado para alguma letra cujo valor ainda não foi determinado.
- Na primeira etapa, geralmente não importa muito em que ordem a propagação é feita, uma vez que todas as propagações disponíveis serão realizadas antes do término da etapa.
- No segundo passo, porém, a ordem em que os palpites são tentados pode ter um impacto substancial no grau de busca que é necessário.

Heurísticas para a adivinhação

- Algumas heurísticas úteis podem ajudar a selecionar o melhor palpite para tentar primeiro.
- Por exemplo, se houver uma letra que tenha apenas dois valores possíveis e outra com seis valores possíveis, há uma chance maior de adivinhar no primeiro que no segundo.
- Outra heurística útil é que, se houver uma letra que participa de muitas restrições, é uma boa ideia preferi-la a uma letra que participa de poucas.
- Um palpite em uma letra tão altamente restrita geralmente levará rapidamente a uma contradição (se estiver errada) ou à geração de muitas restrições adicionais (se estiver certa).
- Um palpite em uma letra menos restrita, por outro lado, fornece menos informações.

Algoritmo Satisfação de Restrições

- O resultado dos primeiros ciclos de processamento deste exemplo é mostrado na Fig. 1.
- Como as restrições nunca desaparecem em níveis mais baixos, somente as que estão sendo adicionadas são mostradas para cada nível.
- Não será muito mais difícil para o solucionador de problemas acessar as restrições como um conjunto de listas do que como uma longa lista, e essa abordagem é eficiente tanto em termos de espaço de armazenamento quanto na facilidade de retrocesso (*backtracking*).
- Outra abordagem razoável para esse problema seria armazenar todas as restrições em uma base de dados central e também registrar em cada nó as alterações que devem ser desfeitas durante o *backtracking*.
- C1, C2, C3 e C4 indicam os bits de transporte (*carry*) das colunas, numerando a partir da direita.

Problema criptaritmético

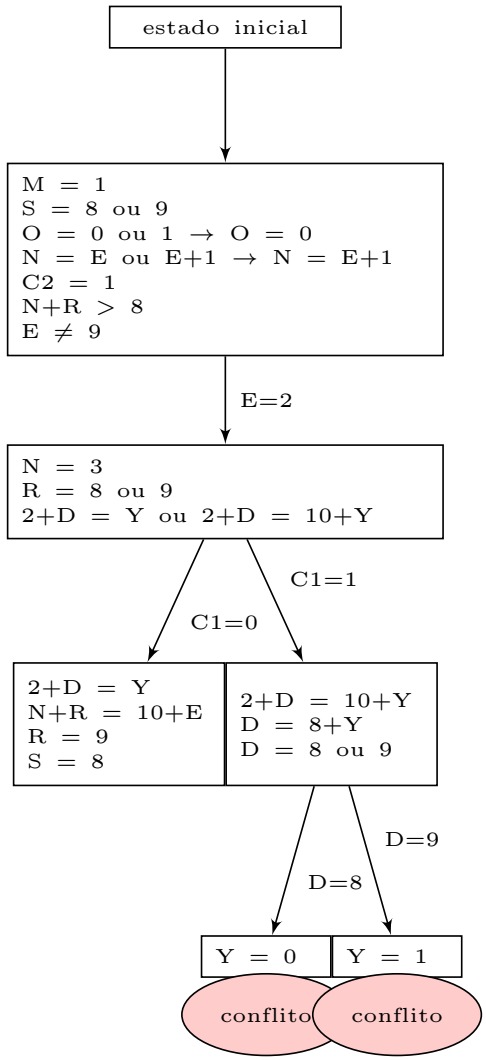


Figure 1: Resolvendo o problema criptaritmético SEND+MORE=MONEY [1].

Problema criptaritmético

- Inicialmente, as regras para propagação de restrições geram as seguintes restrições adicionais:
  - M = 1, já que dois números de dígito único mais um *carry* não podem totalizar mais que 19.
  - S = 8 ou 9, já que S + M + C3 > 9 (para gerar o *carry*) e M = 1, S + 1 + C3 > 9, então S + C3 > 8 e C3 é no máximo 1.
  - O = 0, uma vez que S + M(1) + C3 (<= 1) deve ser pelo menos 10 para gerar um *carry* e pode ser no máximo 11. Mas M já é 1, então O deve ser 0.
  - N = E ou E + 1, dependendo do valor de C2. Mas N não pode ter o mesmo valor que E. Então N = E + 1 e C2 é 1.
  - Para que C2 seja 1, a soma de N + R + C1 deve ser maior que 9, então N + R deve ser maior que 8.
  - N + R não pode ser maior que 18, mesmo com *carry*, portanto, E não pode ser 9.

Problema criptaritmético

- Neste ponto, vamos supor que não possam ser geradas mais restrições.
- Então, para progredir daqui, devemos adivinhar.
- Suponha que a E seja atribuído o valor 2. (Escolhemos adivinhar um valor para E porque ocorre três vezes e, portanto, interage muito com as outras letras.)
- Agora o próximo ciclo começa.

Problema criptaritmético

- O propagador de restrição agora observa que:
  - N = 3, já que N = E + 1.
  - R = 8 ou 9, já que R + N (3) + C1 (1 ou 0) = 2 ou 12. Mas como N já é 3, a soma desses números não-negativos não pode ser menor que 3. Assim, R + 3 + (0 ou 1) = 12 e R = 8 ou 9.
  - 2 + D = Y ou 2 + D = 10 + Y, a partir da soma na coluna mais à direita.

Problema criptaritmético

- Novamente, supondo que nenhuma outra restrição possa ser gerada, é necessário uma adivinhação.
- Suponha que C1 seja escolhido para adivinhar um valor.
- Se tentarmos o valor 1, então, finalmente, alcançaremos becos sem saída, como mostrado na Fig. 1.
- Quando isso acontece, o processo retornará (*backtrack*) e tentará $C1 = 0$.

Problema criptaritmético

- Algumas observações valem a pena neste processo.
- Observe que tudo o que é exigido das regras de propagação de restrição é que elas não infiram restrições espúrias.
- Elas não precisam inferir todas as restrições legais.
- Por exemplo, poderíamos ter raciocinado até o resultado que C1 é igual a 0.
- Poderíamos ter feito isso observando que para C1 ser 1, o seguinte deve valer: $2 + D = 10 + Y$.

Problema criptaritmético

- Para este caso, D teria que ser 8 ou 9.
- Mas ambos S e R devem ser 8 ou 9 e três letras não podem compartilhar dois valores.
- Então C1 não pode ser 1.
- Se tivéssemos percebido isso inicialmente, alguma busca poderia ter sido evitada.
- Mas como que as regras de propagação de restrição usadas não foram tão sofisticadas, custou alguma busca.
- Se a rota de busca leva mais ou menos tempo real que a rota de propagação da restrição depende de quanto tempo leva para executar o raciocínio necessário para a propagação de restrição.

Satisfação de Restrições

- Uma segunda coisa a notar é que geralmente há dois tipos de restrições.
- O primeiro tipo é simples; apenas lista valores possíveis para um único objeto.
- O segundo tipo é mais complexo; descreve relacionamentos entre objetos.
- Ambos os tipos de restrições desempenham o mesmo papel no processo de satisfação de restrições e, no exemplo da criptaritmética, foram tratados de forma idêntica.
- Para alguns problemas, no entanto, pode ser útil representar os dois tipos de restrições de maneira diferente.

Satisfação de Restrições

- As restrições simples de listagem de valores são sempre dinâmicas e, portanto, devem sempre ser representadas explicitamente em cada estado de problema.
- O mais complicado, restrições de expressão de relacionamentos são dinâmicas no domínio criptaritmético, uma vez que são diferentes para cada problema criptaritmético.
- Mas em muitos outros domínios elas são estáticos.
- Sempre que as restrições binárias são estáticas, pode ser computacionalmente eficiente não representá-las explicitamente na descrição do estado mas sim codificá-las diretamente no algoritmo.
- Quando isso é feito, as únicas coisas que são propagadas são valores possíveis.
- Mas o algoritmo essencial é o mesmo em ambos os casos.

Outro exemplo de Criptaritmética

$$\begin{array}{r} \,T\;W\;O \\ +\;T\;W\;O \\ \hline F\;O\;U\;R \end{array}$$

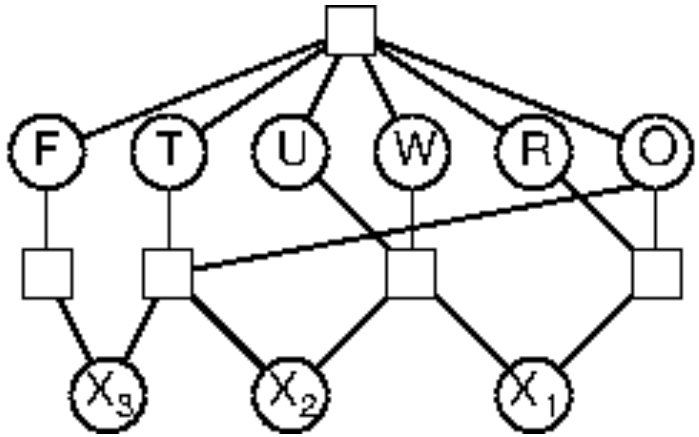


Figure 2: Hipergrafo de restrição: nós comuns (◯) e hipernós (□) que representam restrições n-árias.

- Variáveis: $X = \{F,\ T,\ U,\ W,\ R,\ O,\ X_1,\ X_2,\ X_3\}$,
- Domínios: $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Restrições:
  - $alldiff\,(F, T, U, W, R, O)$
  - $O + O = R + 10 \cdot X_1$, etc.

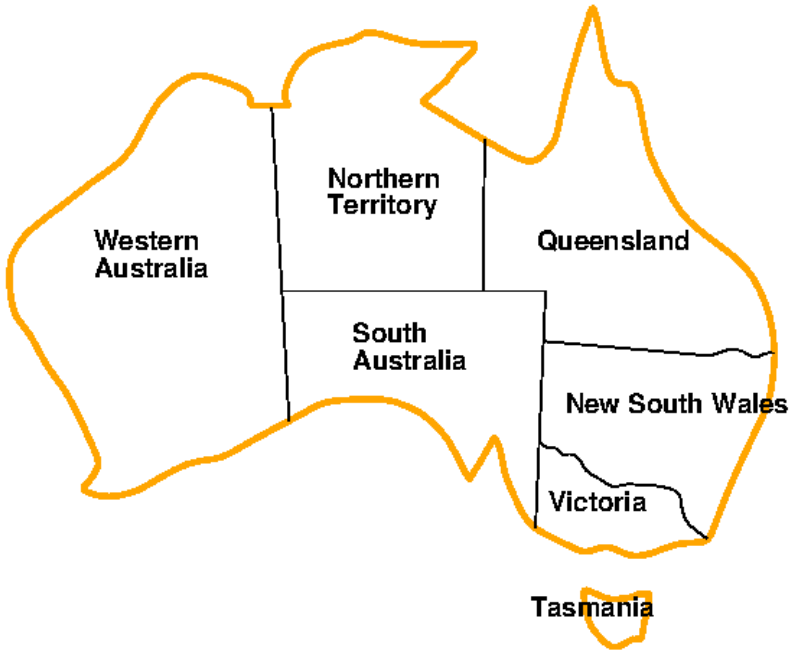
Criptaritmética

- Toda restrição de domínio finito pode ser reduzida a um conjunto de restrições binárias se variáveis auxiliares suficientes forem introduzidas, portanto pode-se transformar qualquer CSP em um com apenas restrições binárias; isto torna os algoritmos mais simples,
- Uma outra forma de converter um CSP *n*-ário em binário é a transformação **grafo dual**: crie um novo grafo no qual há uma variável para cada restrição no grafo original, e uma restrição binária para cada par de restrições no grafo original que compartilham variáveis.

Criptaritmética

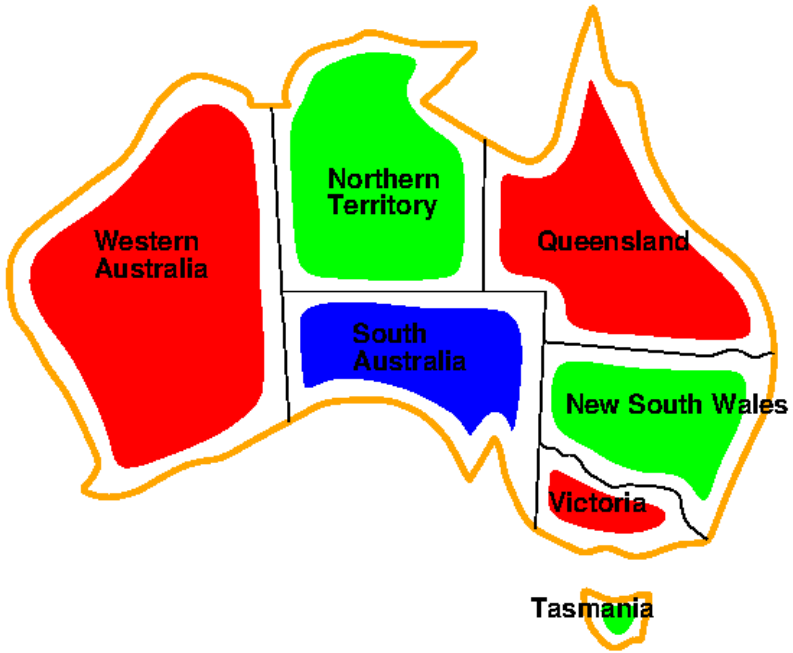
- Por exemplo, se o grafo original tem variáveis $\{X, Y, Z\}$ e restrições $\langle (X, Y, Z), C_1 \rangle$ e $\langle (X, Y), C_2 \rangle$ então o grafo dual teria as variáveis $\{C_1, C_2\}$ com a restrição binária $\langle (X, Y), R_1 \rangle$, onde (X, Y) são as variáveis compartilhadas e R_1 é a nova relação que define a restrição entre as variáveis compartilhadas, como especificado por C_1 e C_2 originais.
- Há no entanto duas razões para preferir uma restrição global como *alldiff* a um conjunto de restrições binárias.
- Primeiro, é mais fácil e menos sujeito a erro escrever a descrição do problema usando *alldiff* .
- Segundo, é possível projetar algoritmos de inferência de propósito especial para restrições globais que não estão disponíveis para um conjunto de restrições mais primitivas.

Coloração de Mapas



- Variáveis: $X = \{WA, NT, Q, NSW, V, SA, T\}$,
- Domínios: $D = \{vermelho, verde, azul\}$,
- Restrições: regiões adjacentes devem ter cores diferentes
  - e.g. $WA \neq NT$ (se a linguagem permite), ou
  - $(WA, NT) \in$
    $\{(vermelho, verde), (vermelho, azul), (verde, vermelho), (verde, azul), ...\}$.

Coloração de Mapas (contd)



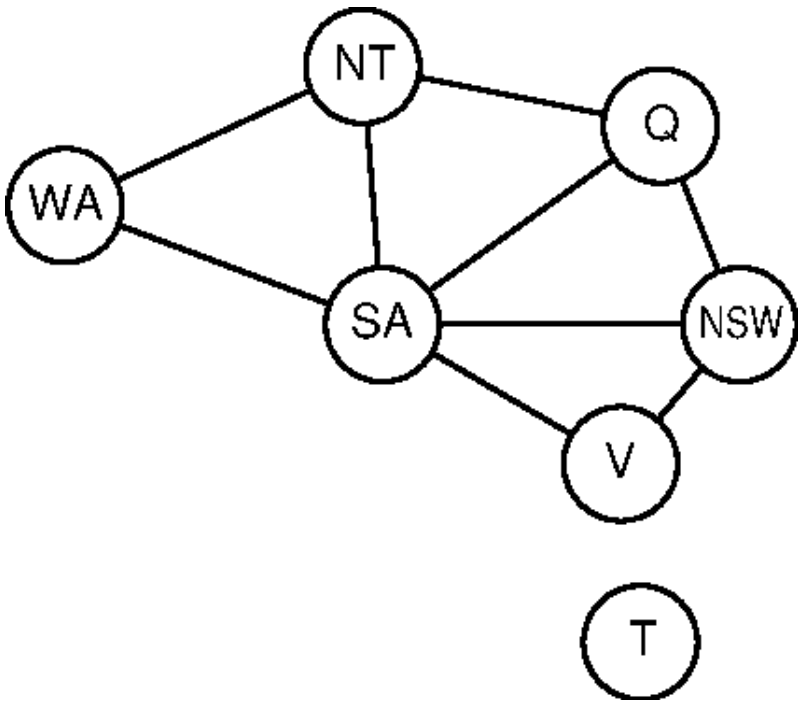
- Soluções são atribuições que satisfazem todas as restrições, e.g. $\{WA = vermelho; NT = verde; Q = vermelho; NSW = verde; V = vermelho; SA = azul; T = verde\}$.

Sumário

1. CSP
   - Introdução
   - Exemplos
   - Grafo de restrição
2. Variações de CSPs
   - Variáveis
   - Propagação de restrições
   - *Backtracking*
3. Estrutura do problema
   - Introdução
   - Conflitos mínimos
   - Resumo

Grafo de restrição

- CSP binário: cada restrição relaciona no máximo duas variáveis,
- **Grafo de restrição**: nós são variáveis, arestas mostram restrições.



Algoritmos CSP de propósito geral usam a estrutura do grafo para acelerar a busca. Por exemplo, a Tasmânia é um subproblema independente!

Sumário

1 CSP
- Introdução
- Exemplos
- Grafo de restrição

2 Variações de CSPs
- Variáveis
- Propagação de restrições
- *Backtracking*

3 Estrutura do problema
- Introdução
- Conflitos mínimos
- Resumo

Variações de CSPs

- Variáveis discretas
  - domínios finitos; tamanho $d \Rightarrow \mathcal{O}(d^n)$ tarefas completas
    - e.g. CSPs booleanos, incluindo satisfazibilidade booleana (NP-completo) em domínios infinitos (inteiros, strings, etc.),
    - e.g. agendamento de trabalho, as variáveis são dias de início/término para cada trabalho,
    - precisa de uma linguagem de restrição, e.g. $StartJob_1 + 5 \leq StartJob_3$,
    - restrições lineares solucionáveis, não lineares indecidíveis.
- Variáveis contínuas
  - e.g. horários de início/fim das observações do Telescópio Hubble,
  - restrições lineares solúveis em tempo polinomial por métodos de programação linear[1].

[1] Em matemática, problemas de **Programação Linear** (PL) são problemas de otimização nos quais a função objetivo e as restrições são todas lineares.

Variações de restrições

- Restrições **unárias** envolvem uma única variável, e.g. $SA \neq verde$,
- Restrições **binárias** envolvem pares de variáveis, e.g. $SA \neq WA$,
- Restrições de **ordem superior** envolvem 3 ou mais variáveis, e.g. restrições de colunas criptaritméticas,
- **Preferências** (restrições leves), e.g. *vermelho* é melhor que *verde* muitas vezes representável por um custo para cada atribuição de variável $\rightarrow$ problemas de otimização restritos.

Sumário

1. CSP
   - Introdução
   - Exemplos
   - Grafo de restrição
2. Variações de CSPs
   - Variáveis
   - Propagação de restrições
   - *Backtracking*
3. Estrutura do problema
   - Introdução
   - Conflitos mínimos
   - Resumo

Propagação de Restrições: Inferência em CSPs

- Na busca regular de espaços de estado, um algoritmo pode fazer apenas uma coisa: busca!
- Em CSPs há a busca (escolher uma nova atribuição de variáveis a partir de várias possibilidades) ou fazer um tipo específico de **inferência** chamado de **propagação de restrições**: usando as restrições para reduzir o número de valores legais para uma variável, que por sua vez, pode reduzir os valores legais para outra variável, etc.

Formulação de busca padrão (incremental)

- Vamos começar com a abordagem direta e ingênua, depois a ajustamos,
- Estados são definidos pelos valores atribuídos até agora:
  - Estado inicial: a atribuição vazia, $\{\}$,
  - Função sucessora: atribua um valor a uma variável não atribuída que não conflita com atribuição atual $\Rightarrow$ falha se não houver atribuições legais (não ajustável!)
- Teste de meta: a tarefa atual está concluída.

Formulação de busca padrão (incremental)

1. Isso é o mesmo para todos os CSPs! ☺
2. Toda solução aparece na profundidade n com n variáveis $\Rightarrow$ use a busca em profundidade,
3. O caminho é irrelevante, então também pode usar a formulação de estado completo,
4. $b = (n - l)d$ na profundidade l, daí $n!d^n$ folhas!!!! ☹
   - b = fator de ramificação
   - d = tamanho do domínio

Por que formular um problema como um CSP?

- Uma razão é que os CSPs fornecem uma representação natural para uma grande variedade de problemas,
- Os solucionadores CSP podem ser mais rápidos que buscadores de espaço de estados, pois os solucionadores CSP podem eliminar grandes porções do espaço de busca,
- No problema da Austrália, uma vez que escolhemos $\{SA = azul\}$, pode-se concluir que nenhuma das cinco variáveis vizinhas podem assumir o valor *azul*,
- Sem levar vantagem da propagação de restrições, um procedimento de busca teria de considerar $3^5 = 243$ atribuições para as cinco variáveis vizinhas; com a propagação de restrições nunca teríamos de considerar *azul* como um valor, portanto teríamos apenas $2^5 = 32$ atribuições para buscar, uma redução de 87%.

Por que formular um problema como um CSP?

- Na busca regular de espaços de estado, pode-se perguntar apenas: este estado específico é uma meta? Não? E este aqui?
- Com CSPs, uma vez descoberto que uma atribuição parcial não é uma solução, pode-se imediatamente descartar refinamentos adicionais da atribuição parcial,
- Além disso, pode-se ver *porque* a atribuição não é uma solução - vemos quais variáveis violam uma restrição - portanto podemos focar a atenção nas variáveis que importam,
- Como resultado, muitos problemas que são intratáveis para a busca regular de espaços de estado podem ser resolvidos rapidamente quando formulados como CSPs.

Sumário

1 CSP
- Introdução
- Exemplos
- Grafo de restrição

2 Variações de CSPs
- Variáveis
- Propagação de restrições
- *Backtracking*

3 Estrutura do problema
- Introdução
- Conflitos mínimos
- Resumo

Busca *backtracking*

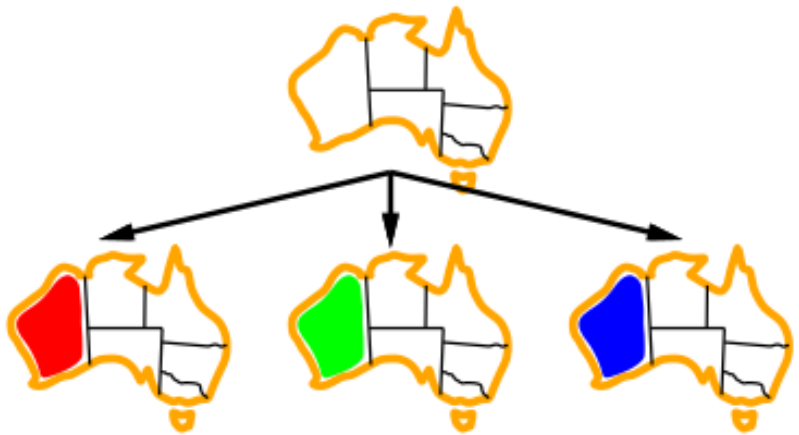
- Atribuições de variáveis são comutativas, i.e. [$WA = vermelho$ então $NT = verde$], o mesmo que [$NT = verde$ então $WA = vermelho$],
- Só precisa considerar as atribuições para uma única variável em cada nó $\Rightarrow b = d$ e há d^n folhas,
- Busca em profundidade por CSPs com atribuições de variável única é chamada de **busca backtracking**,
- A busca *backtracking* é o algoritmo básico não informado para CSPs,
- Pode resolver o problema das n rainhas para $n \approx 25$.

Busca *backtracking*

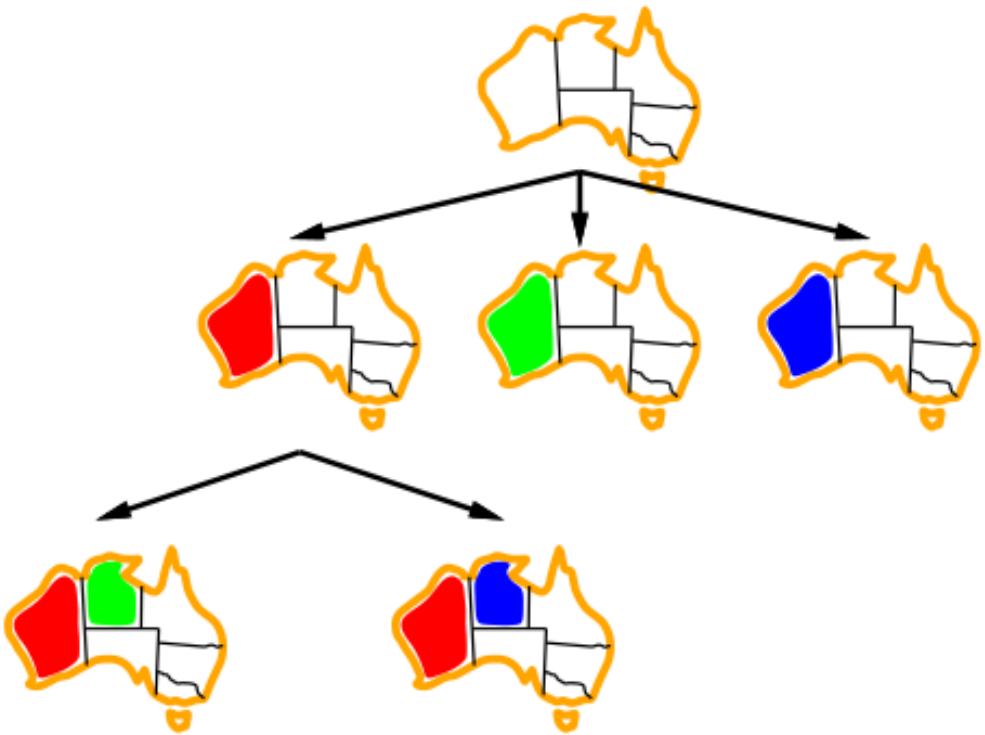
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Exemplo de *backtracking*

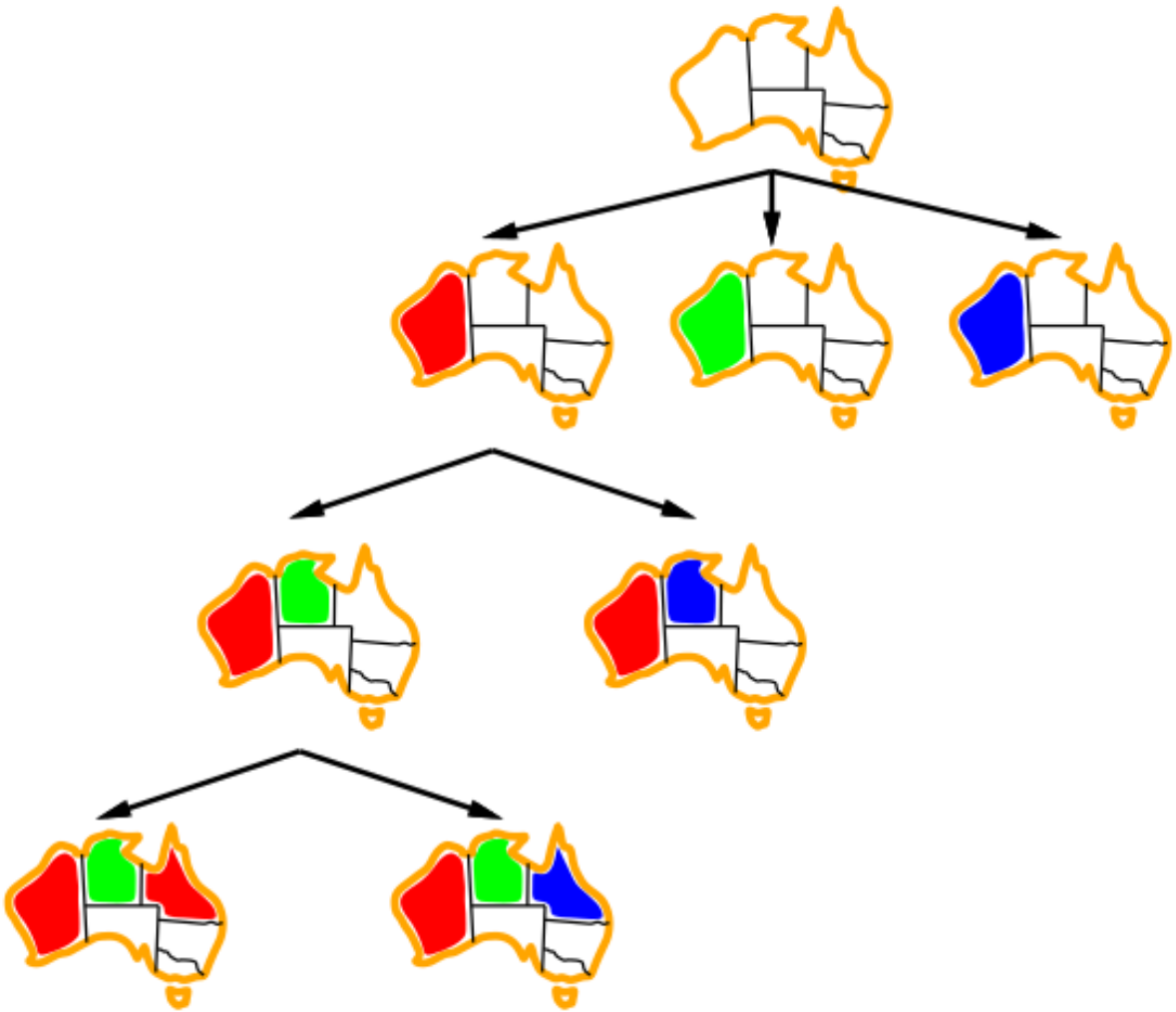
Exemplo de *backtracking*



Exemplo de *backtracking*



Exemplo de *backtracking*

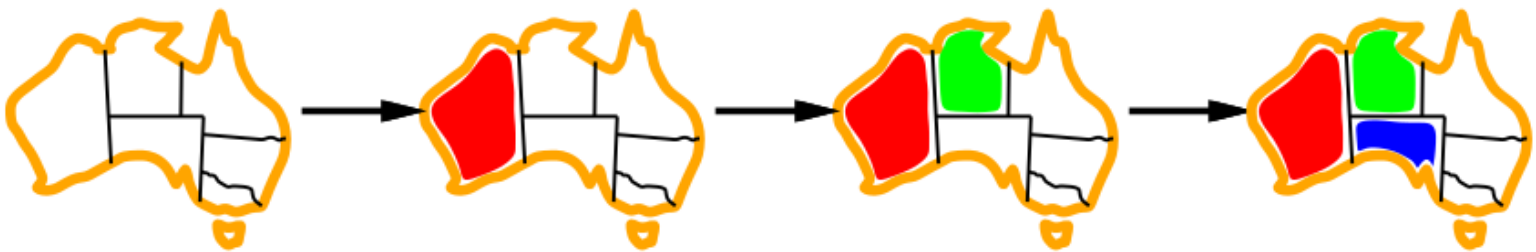


Melhorando a eficiência do *backtracking*

- Métodos de **propósito geral** podem dar grandes ganhos em velocidade:
  1. Qual variável deve ser atribuída em seguida?
  2. Em que ordem seus valores devem ser tentados?
  3. Podemos detectar falhas inevitáveis precocemente?
  4. Podemos tirar vantagem da estrutura do problema?

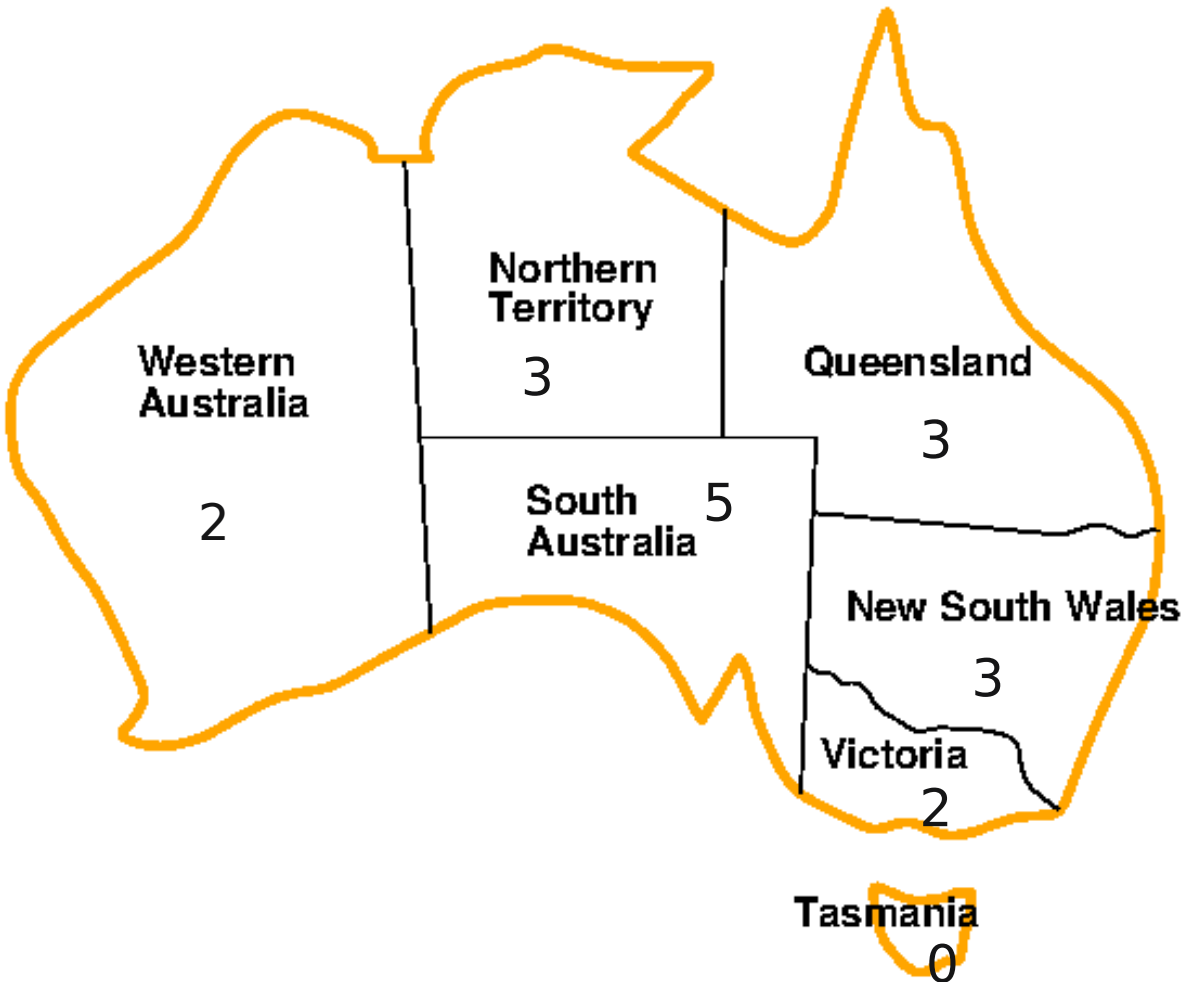
H1: Valores residuais mínimos (MRV)

- Valores residuais mínimos (MRV): escolha a variável com o menor número de valores legais.



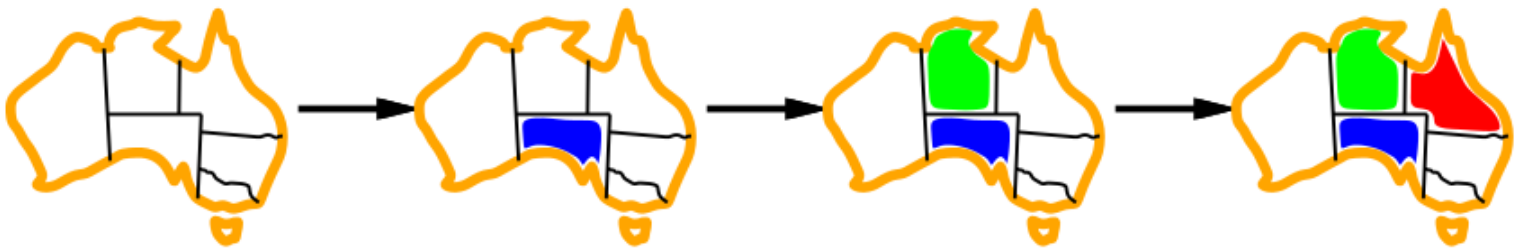
H2: Heurística de grau

- Grau: número de restrições.



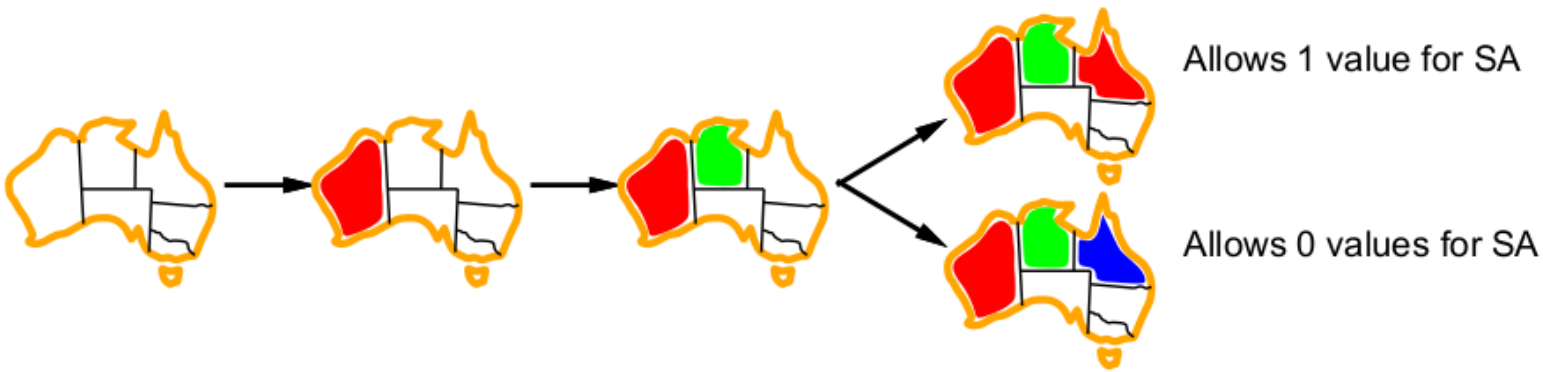
H2: Heurística de grau

- Desempate entre variáveis do MRV,
- Heurística de grau: escolha a variável com mais restrições nas variáveis restantes (maior grau).



H3: Menor valor de restrição

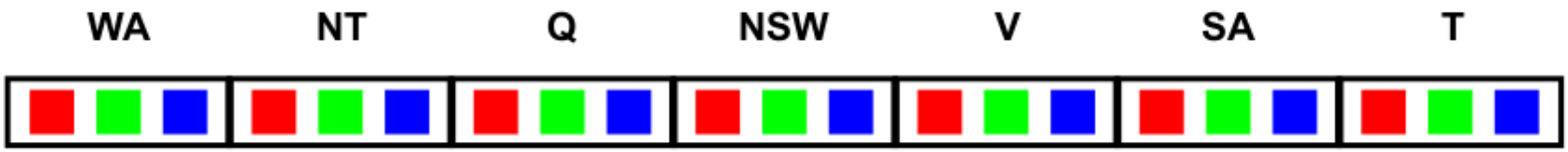
- Dada uma variável, escolha o menor valor de restrição: aquele que exclui o menor número de variáveis restantes.



- Combinar essas heurísticas (H1, H2 e H3) faz com que 1000-rainhas seja viável.

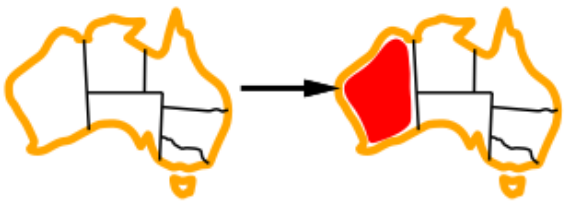
Verificação progressiva

- Idéia: acompanhe os valores legais restantes para variáveis não atribuídas. Terminar a busca quando nenhuma variável tiver valores legais.



Verificação progressiva

- Idéia: acompanhe os valores legais restantes para variáveis não atribuídas. Terminar a busca quando nenhuma variável tiver valores legais.

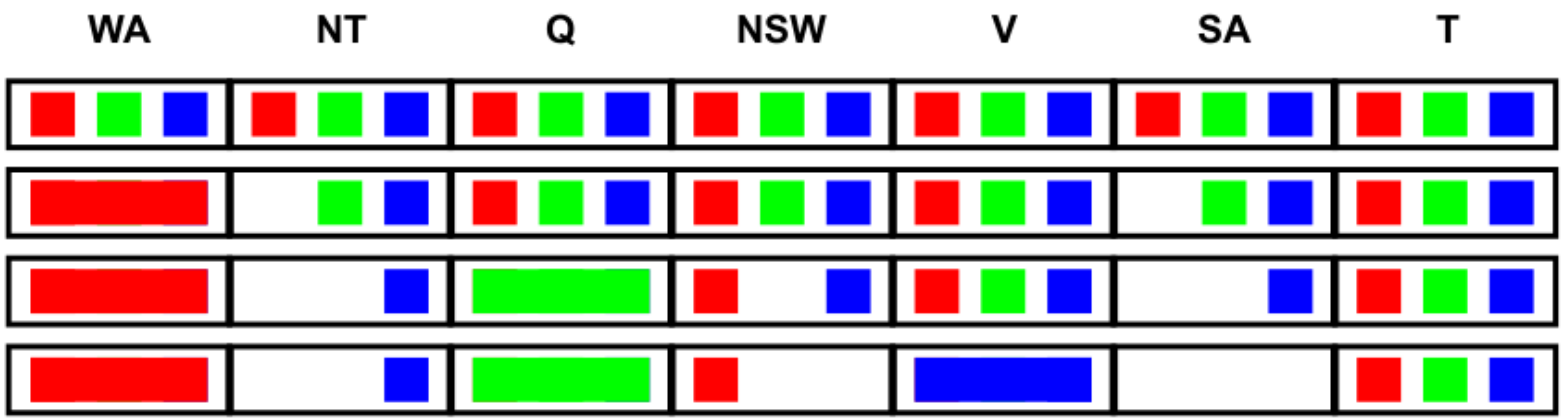


Verificação progressiva

- Idéia: acompanhe os valores legais restantes para variáveis não atribuídas. Terminar a busca quando nenhuma variável tiver valores legais.

Verificação progressiva

- Idéia: acompanhe os valores legais restantes para variáveis não atribuídas. Terminar a busca quando nenhuma variável tiver valores legais.



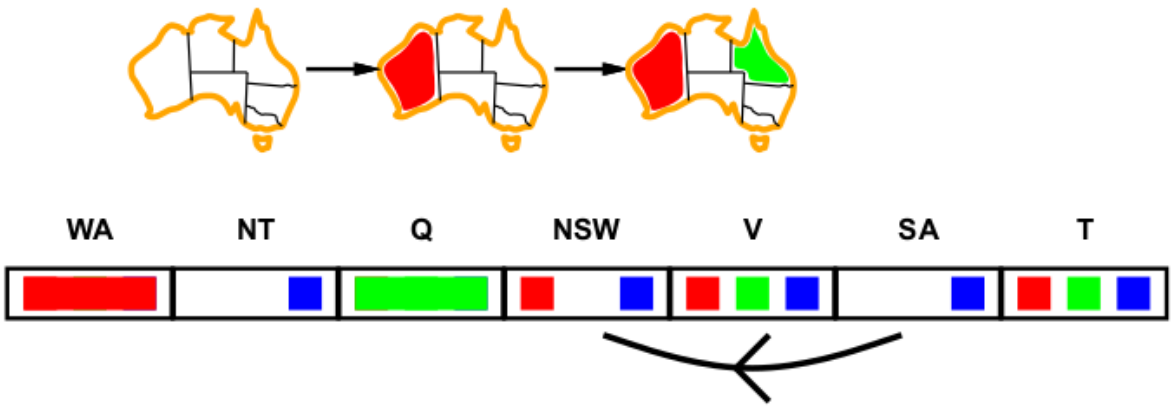
Propagação de restrições

- A verificação progressiva propaga as informações das variáveis atribuídas às variáveis não atribuídas, mas não fornece detecção antecipada para todas as falhas:

- NT e SA não podem ser ambos azuis!
- Propagação de restrições repetidamente impõe restrições localmente.

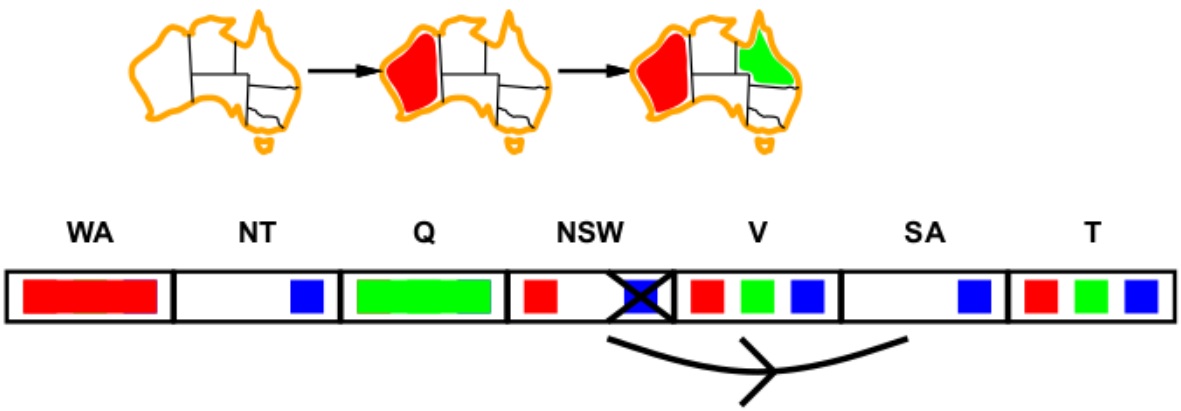
Consistência da aresta

- A forma mais simples de propagação torna cada aresta consistente,
- $X \rightarrow Y$ é consistente sse para cada valor x de X há algum y permitido.



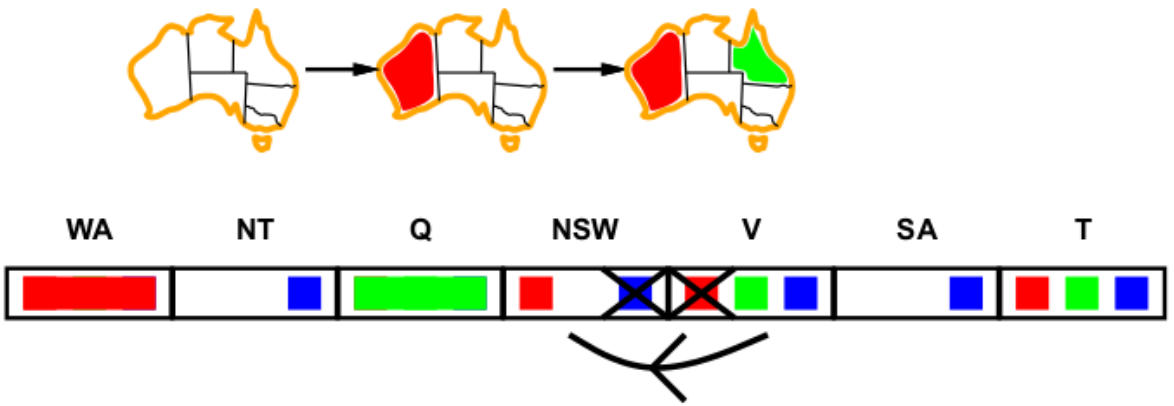
Consistência da aresta

- A forma mais simples de propagação torna cada aresta consistente,
- $X \rightarrow Y$ é consistente sse para cada valor x de X há algum y permitido.



Consistência da aresta

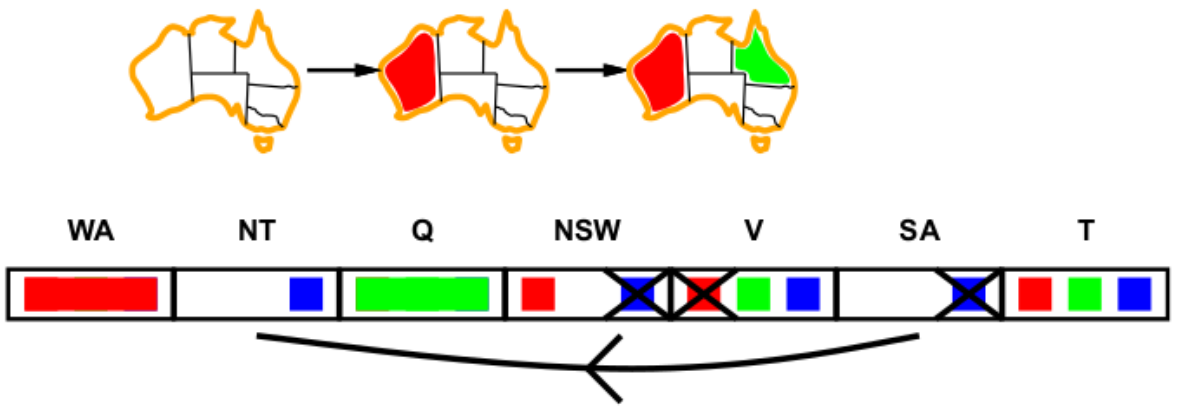
- A forma mais simples de propagação torna cada aresta consistente,
- $X \rightarrow Y$ é consistente sse para cada valor x de X há algum y permitido.



- Se X perde um valor, vizinhos de X precisam ser checados novamente.

Consistência da aresta

- A forma mais simples de propagação torna cada aresta consistente,
- $X \rightarrow Y$ é consistente sse para cada valor x de X há algum y permitido.



- Se X perde um valor, vizinhos de X precisam ser checados novamente,
- A consistência da aresta detecta falha antes da verificação progressiva,
- Pode ser executado como um pré-processador ou após cada atribuição.

Algoritmo consistência da aresta

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS$[X_i]$ **do**
        add (X_k, X_i) to *queue*

---

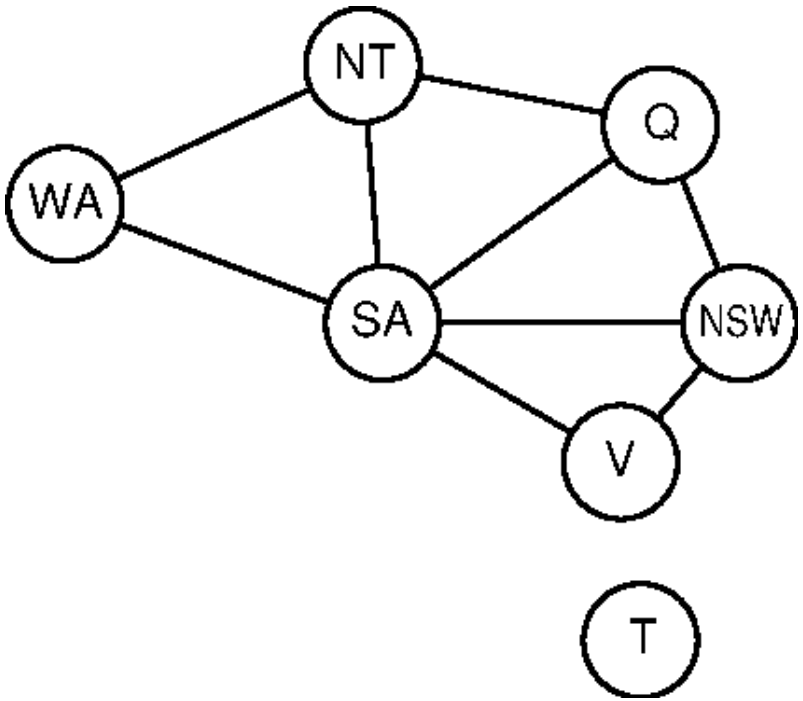
function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff succeeds
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN$[X_i]$ **do**
    if no value y in DOMAIN$[X_j]$ allows (x, y) to satisfy the constraint $X_i \leftrightarrow X_j$
      then delete x from DOMAIN$[X_i]$; *removed* $\leftarrow$ *true*
  return *removed*

$\mathcal{O}(n^2 d^3)$, pode ser reduzido a $\mathcal{O}(n^2 d^2)$ (mas detectar **todos** é NP-difícil)

Sumário

1. CSP
   - Introdução
   - Exemplos
   - Grafo de restrição
2. Variações de CSPs
   - Variáveis
   - Propagação de restrições
   - *Backtracking*
3. Estrutura do problema
   - Introdução
   - Conflitos mínimos
   - Resumo

Estrutura do problema

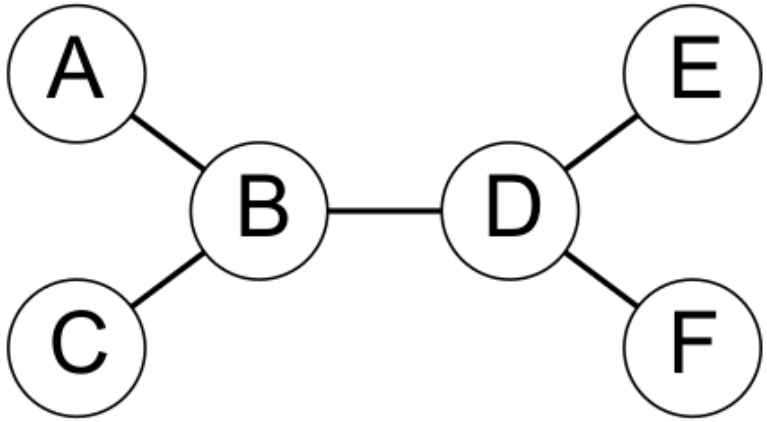


Tasmânia e continente são **subproblemas independentes**, Identificáveis como **componentes conectados** do grafo de restrição.

Estrutura do problema

- Suponha que cada subproblema tenha c variáveis de um total de n,
- O custo da solução do pior caso é $n/c \cdot d^c$, **linear** em n (n/c é o número de subproblemas, d é o tamanho do domínio) $\Rightarrow$ sem a decomposição: $\mathcal{O}(d^n)$, exponencial em n!
- E.g. $n = 80$, $d = 2$, $c = 20$:
  - $2^{80} = 4$ bilhões de anos a 10 milhões de nós/seg,
  - $4 \cdot 2^{20} = 0,4$ segundos a 10 milhões de nós/seg.

CSPs estruturados em árvore



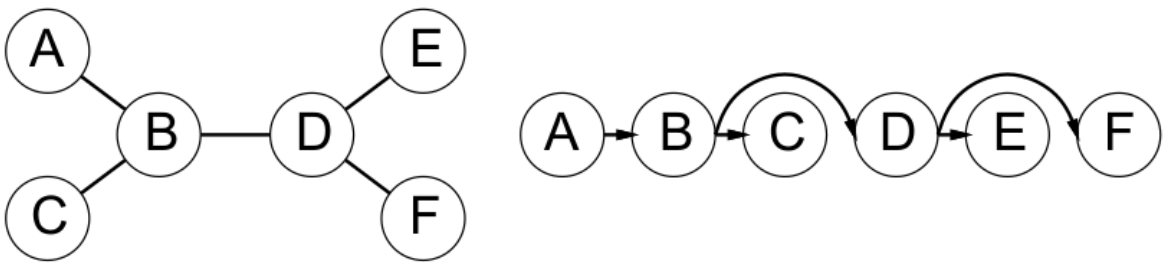
Theorem

se o grafo de restrição não tem loops, o CSP pode ser resolvido no tempo $\mathcal{O}(nd^2)$

- Compare com os CSPs gerais, onde o pior caso é $\mathcal{O}(d^n)$,
- Esta propriedade também se aplica ao raciocínio lógico e probabilístico: um exemplo importante da relação entre restrições sintáticas e a complexidade do raciocínio.

Algoritmo para CSPs estruturados em árvore

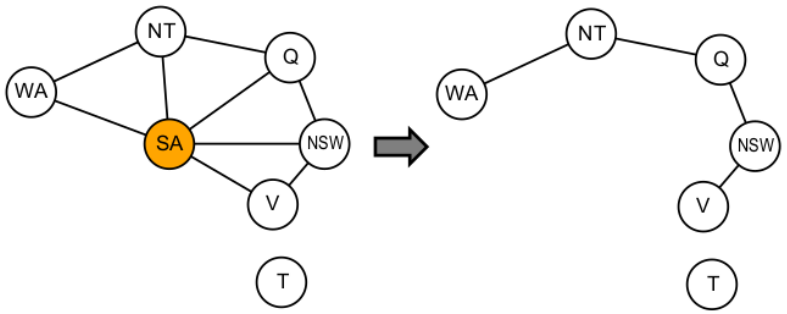
1. Escolha uma variável como raiz, ordene variáveis da raiz às folhas tal que o pai de cada nó preceda-o na ordenação



2. Para j de n até 2, aplique $RemoveInconsistent(Parent(X_j), X_j)$
3. Para j de 1 a n, atribua X_j consistentemente com $Parent(X_j)$

CSPs estruturados em quase-árvore

- **Condicionamento**: instanciar uma variável, podar os domínios de seus vizinhos



- Condicionamento *cutset*[2]: instanciar (em todos os caminhos) um conjunto de variáveis tal que o grafo de restrição restante é uma árvore
- Tamanho do *cutset* $c \Rightarrow$ tempo de execução $\mathcal{O}(d^c \cdot (n-c)d^2)$, muito rápido para c pequeno

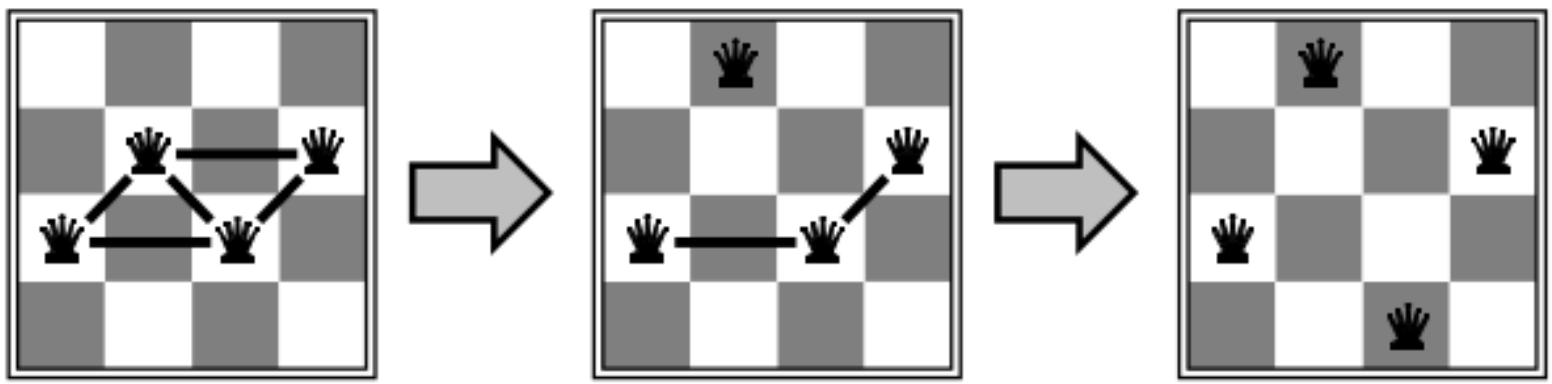
[2] Na teoria dos grafos, um corte (*cut*) é uma partição dos vértices de um gráfico em dois subconjuntos disjuntos. Qualquer corte determina um conjunto de cortes (*cutset*), o conjunto de arestas (arcos) que têm um ponto final em cada subconjunto da partição. Diz-se que estas arestas atravessam o corte. Em um grafo conectado, cada conjunto de corte determina um corte exclusivo e, em alguns casos, os cortes são identificados com seus conjuntos de corte, e não com suas partições de vértice.

Algoritmos iterativos para CSPs

- Subida de encosta (*Hill-Climbing*), recozimento simulado (*simulated annealing*) normalmente trabalham com estados "completos", isto é, todas as variáveis atribuídas
- Para aplicar aos CSPs:
  - permitir estados com restrições insatisfeitas
  - operadores **reatribuem** valores de variáveis
- Seleção de variáveis: selecione aleatoriamente qualquer variável com conflito
- Seleção de valor pela heurística de conflitos mínimos:
  - escolha um valor que viole o menor número de restrições
  - isto é, *hillclimb* com $h(n) =$ número total de restrições violadas

Exemplo: 4 rainhas

- **Estados**: 4 rainhas em 4 colunas ($4^4 = 256$ estados)
- **Operadores**: mova a rainha na coluna
- **Teste de meta**: nenhum ataque
- **Avaliação**: $h(n)$ = número de ataques



Sumário

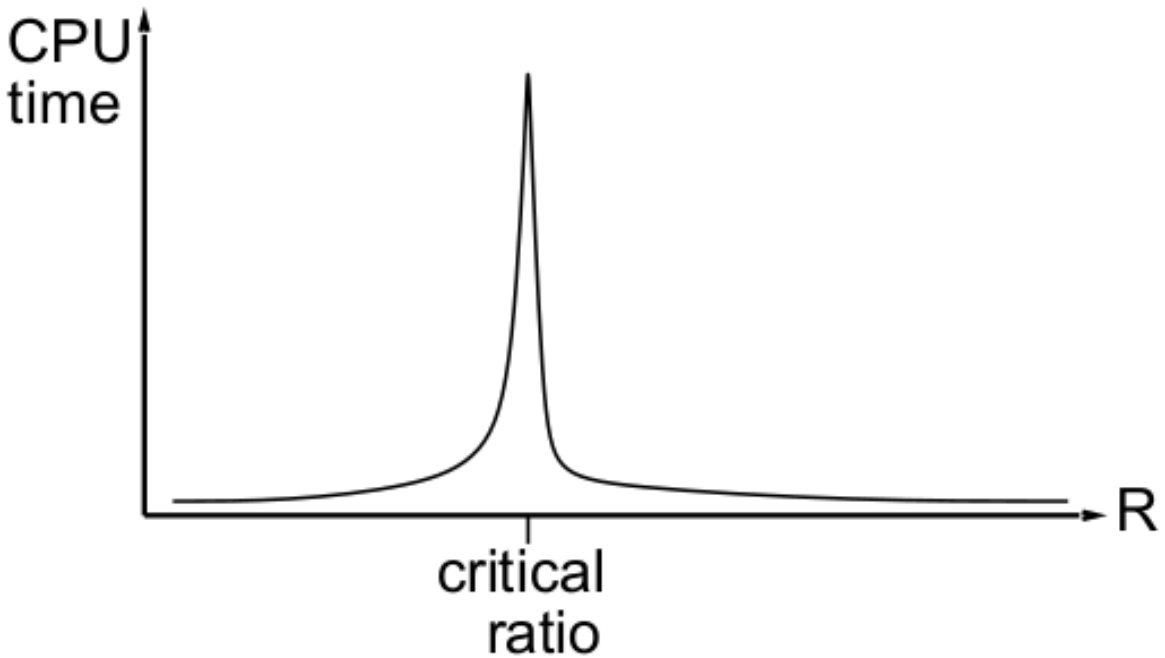
1 CSP
- Introdução
- Exemplos
- Grafo de restrição

2 Variações de CSPs
- Variáveis
- Propagação de restrições
- *Backtracking*

3 Estrutura do problema
- Introdução
- Conflitos mínimos
- Resumo

Performance da heurística de conflitos mínimos

- Dado estado inicial aleatório, pode-se resolver n-rainhas em tempo quase constante para n arbitrário com alta probabilidade (e.g. $n = 10.000.000$),
- O mesmo parece ser verdade para qualquer CSP gerado aleatoriamente **exceto** em uma faixa estreita da relação $R =$ número de restrições/número de variáveis.



Sumário

1 CSP
- Introdução
- Exemplos
- Grafo de restrição

2 Variações de CSPs
- Variáveis
- Propagação de restrições
- *Backtracking*

3 Estrutura do problema
- Introdução
- Conflitos mínimos
- Resumo

Resumo

- Os CSPs são um tipo especial de problema:
  - estados definidos por valores de um conjunto fixo de variáveis
  - teste de meta definido por **restrições** em valores de variáveis
- *Backtracking* = busca em profundidade com uma variável atribuída por nó
- Ordenação de variável e heurística de seleção de valor ajudam significativamente
- A verificação progressiva impede atribuições que garantam falhas posteriores

Resumo

- Propagação de restrição (por exemplo, consistência da aresta) faz um trabalho adicional para restringir valores e detectar inconsistências
- A representação do CSP permite a análise da estrutura do problema
- CSPs estruturados em árvore podem ser resolvidos em tempo linear
- Os conflitos mínimos iterativos geralmente são eficazes na prática

CSPs do mundo real

- Problemas de atribuição: e.g. quem ensina que classe,
- Problemas de horários: e.g. qual classe é oferecida quando e onde,
- Configuração de hardware,
- Planilhas,
- Agendamento de transporte,
- Agendamento de fábrica,
- Representação esquemática de CI (*floorplanning*).

Observe que muitos problemas do mundo real envolvem variáveis reais.

Referências I

[1] Rich, E., Knight, K.
Artificial Intelligence, 2nd. edition.
McGraw-Hill, 1991.

[2] Russell, S., Norvig, P.
Artificial Intelligence - A Modern Approach. 3rd. edition.
Prentice Hall, 2010.